PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES

DISEÑO E IMPLEMENTACIÓN DEL JUEGO SOKOBAN PARA PC MEDIANTE EL MOTOR 3D GAME STUDIO

Autor: Alejandro Aguilar León

Tutor: Juan Peralta Donate

Leganés, Julio de 2011

Título: Diseño e implementación del juego sokoban para PC mediante el motor 3d Game Studio

Autor: Alejandro Aguilar León

Director: Juan Peralta Donante

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del proyecto fin de carrera el día ___ de _____ de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

- A mi mujer que me ha ayudado con mis hijas y el trabajo para poder tener tiempo para desarrollar este proyecto.
- A D. Juan Peralta Donate. Por haber entendido perfectamente mis circunstancias y haber adaptado el proyecto a estas y así poder llevarlo a cabo.

Resumen

A día de hoy es difícil imaginar un hogar donde no haya un ordenador o una videoconsola, de ahí que la industria de los videojuegos se haya convertido en un negocio que mueve millones de euros, sólo en España en 2007 facturó 1454 millones de euros. Según un estudio de Paul Heydon {1} de la empresa Avisa Partnes, este sector genera 105.000 millones de dólares al año en todo el mundo {2}**¡Error! No se encuentra el origen de la referencia.**.

El mundo de los videojuegos está en contaste evolución tanto en las plataformas que los desarrollan como en las máquinas que los soportan, de ahí que hayan surgido multitud de motores gráficos {3} para su desarrollo. Este proyecto va a desarrollar un juego en 3D con uno de estos motores disponibles,3D Game Studio A7 {4} . Se usará su versión gratuita y libre.

El juego será una adaptación de un clásico de los años 80, llamado Sokoban {5} . Es un juego de lógica, en el que el jugador controla un personaje que tiene como objetivo colocar en las posiciones marcadas una serie de cajas que se encuentran distribuidas a lo largo de todo el nivel. La dificultad se encuentra en que los movimientos son limitados porque solo puede empujar las cajas y no tirar de ellas, con lo que si coloca una caja en una esquina no podrá sacarla de ahí.

Palabras clave: 3D Game Studio A7, 3D, Sokoban , videojuego.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Código

Capítulo 1:

Introducción

En este capítulo además de la introducción del proyecto se analizan los objetivos marcados de inicio, las diferentes fases del desarrollo del proyecto y un resumen de cómo está organizada la memoria.

1.1 Introducción

A día de hoy es difícil imaginar un hogar donde no haya un ordenador o una videoconsola, de ahí que la industria de los videojuegos se haya convertido en un negocio que mueve millones de euros, sólo en España en 2007 facturó 1454 millones de euros. Según un estudio de Paul Heydon {1} de la empresa Avisa Partnes, este sector genera 105.000 millones de dólares al año en todo el mundo {2}**¡Error! No se encuentra el origen de la referencia.**.

El mundo de los videojuegos está en contaste evolución, tanto en las plataformas que los desarrollan como en las máquinas que los soportan. Parecen que han pasado siglos desde que salió al mercado el primer juego de la historia, pero sólo han pasado 49 años. Observando la figura 1 que tiene dos capturas, una del juego Pong de 1972 y la otra de Killzone3 de 2011, se puede hacer uno a la idea de que se está hablando.

De este punto surge la idea para este proyecto, desarrollar un videojuego con la herramientas disponibles en el mercado actual. En este proyecto se va a ver cómo de manera

sencilla y sin tener grandes conocimientos de programación se puede llegar a conseguir crear un videojuego. Para ello la herramienta elegida es 3D Game Studio A7 {4}.

Figura 1: Pong (1972) y Killzone 3 (2011) .

3D Game Studio A7 es una aplicación para la producción de juegos en dos y tres dimensiones. El núcleo del programa se denomina A7 (actualmente ya está disponible la versión A8), es un motor gráfico que controla la imagen y el comportamiento del mundo virtual. Trabaja de igual forma con espacios interiores y exteriores. También soporta sombras estáticas y dinámicas, estas se mueven en relación a la fuente de la luz y cambia en tiempo real.

A su motor 3D se le unen otras herramientas que complementan el programa y que permiten la producción de videojuegos de gran calidad gráfica tales como motor de efectos y partículas, motor de física y colisiones, motor bidimensional, motor sonoro, motor de red, etcétera.

Si bien es cierto que la aplicación permite desarrollar un juego sin tener conocimientos de programación, quienes deseen profundizar más en el programa y crear aplicaciones más complejas, Game Studio emplea Lite-c, una versión simplificada de C++.

El juego que se ha elegido para desarrollar es un clásico de los años 80, llamado Sokoban. En 1984 Spectrum HoloByte {6} lo saco al mercado europeo. La dinámica de este es sencilla, consiste en un escenario en que hay una serie de cajas descolocadas y el personaje se tiene que encargar de colocarlas en las zonas marcadas. De ahí que se llame Sokoban, termino japonés que significa Warhouse Man, lo que se puede traducir como encargado de almacén. La dificultad radica en que las cajas no se pueden mover de cualquier manera, sólo se pueden empujar, nunca tirar de ellas con lo que si se pega a un muro no las podremos sacar de ahí.

El motivo de la elección de Sokoban, es que no plantea una dinámica de juego complicada con lo que permitirá centrarse más en el desarrollo que en el análisis del juego en sí.

En la figura 2 se muestran un par de capturas del sokoban original, evidentemente la calidad gráfica era la que podían dar los sistemas de la época, pero se pude apreciar claramente en qué consiste el juego.

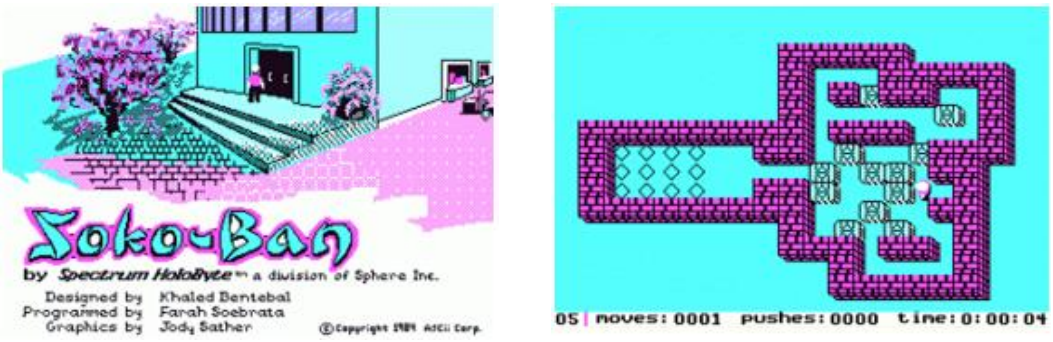

Figura 2: Imágenes de Sokoban de 1984.

Finalmente en la figura 3 se puede ver un par de capturas del resultado final de la aplicación.

Figura 3: Imágenes del resultado final del proyecto.

1.2 Objetivos

Este proyecto tiene marcado varios objetivos que cumplir, que se enumeran a continuación:

El objetivo principal es desarrollar un videojuego sencillo que sirva de ejemplo práctico para todo el que se quiera iniciar en el uso de este aplicación.

Dada la escasez de ejemplos y sitios en internet en los que se pueden llegar a aprender a usar esta herramienta, un segundo objetivo es conseguir explicar de una manera clara y práctica cómo usar esta aplicación.

Como todo proyecto, se pretende dejar una documentación detallada, completa y entendible del este, que sea útil para futuros trabajados relacionados con la materia..

Y como objetivo final, dado que hay una gran variedad de motores gráficos disponibles se quiere valorar si esta herramienta es útil frente a sus competidores.

1.3 Fases del Desarrollo

Con la finalidad de cumplir los objetivos marcados en el anterior punto y acometerlos de manera organizada y práctica se ha divido el proyecto en las siguientes fases:

1. **Documentación y estudio de la aplicación**: Primeramente antes de empezar el proyecto ha habido una fase de documentarse sobre la aplicación, y aprendizaje del manejo de esta. Se han revisado los tutoriales {7} y manual {8}, disponibles en la página web de la propia aplicación.
2. **Toma de requisitos**: Una vez dominado el motor 3D Game Studio, se ha analizado la aplicación desde el punto de vista de la jugabilidad, qué opciones y controles debería tener. Además se han recopilado los requisitos que pedía el cliente, que en este caso es el propio tutor del proyecto.
3. **Diseño del juego:** A partir de la fase de análisis anterior, se diseña a todos los niveles los componentes necesarios de la aplicación para posteriormente poder llegarla a implementar. Se indica cómo debe ser el interfaz que proporciona al usuario a la hora de navegar por los diferentes menús. Un diseño sencillo de cómo serán los niveles, los menús, personajes y pantallas.
4. **Implementación**: En este punto es dónde se concentra el grueso del proyecto, en esta fase se lleva a la realidad todo lo que se había analizado y diseñado en los pasos anteriores. Aquí al tratarse de un videojuego, habrá que crear por un lado los jugadores, por otro lado los niveles, el interfaz del juego y finalmente todo el código para la ejecución de la aplicación.
5. **Fase de pruebas y jugabilidad**: Una vez implementado el juego se ha probado por diferentes usuarios para determinar posibles mejoras de jugabilidad y diseño, para valorar si es necesario volver a fases anteriores y mejorar el resultado final.

1.4 Medios Empleados

Como toda aplicación informática, hemos necesitado tanto medios físicos (Hardware) y medios lógicos (Software), pero en todo momento se ha buscado que los medios empleados sean de acceso libre y gratuito.

- **Medios físicos (Hardware):**
  - Se ha utilizado un PC Intel Core2 Quad de 2,50GHZ, con una memoria RAM de 4 GB y con una tarjeta gráfica de gama baja Nvidia GeForce G210.
- **Medios Lógicos (Software):**
  - Sistema Operativo Windows7 Home Premium. Ya que el motor gráfico sólo está disponible para sistemas operativos Windows.
  - 3D Game Studio A7, se ha descargado e instalado la versión gratuita, para la implementación del juego {9}.
  - Paint, es una aplicación gratuita disponible con el propio Windows. Se ha utilizado como editor de imágenes.
  - eezPix, es una aplicación gratuita de descarga libre de internet {10}. Se trata otro editor de imágenes, que permite trabajar con imágenes de extensión pcx.
  - Office 2007 para la documentación del proyecto. De este paquete de Microsoft se han usado Word, Excel y PowerPoint.

1.5 Estructura de la memoria

La memoria del proyecto está compuesta de siete capítulos, a continuación se resume qué contiene cada uno de ellos:

En el primer capítulo, "*Introducción*", se da una idea de cuál es el origen del proyecto, lo que va a tratar, cuáles son sus objetivos y los medios empleados para su realización.

En el segundo capítulo, que lleva por tema, "*Estado de la cuestión*", se hace un breve resumen de la evolución de los videojuegos a lo largo de los últimos 70 años que tienen de

vida y se mencionan algunos ejemplos de aplicaciones disponibles en la actualidad para el desarrollo de videojuegos y las razones por las que se ha decidido finalmente utilizar 3D Game Studio.

En el tercer capítulo, "*Análisis, diseño, planificación e implementación*", se explica cómo se han llevado a cabo todas las fases del proyecto.

En el cuarto capítulo, "*Planificación y presupuestos iniciales*", se presenta una estimación del tiempo que llevará el desarrollo del proyecto y los costes que tendrá.

En el quinto capítulo, "C*onclusiones*", se analizan la consecución de los objetivos marcados de inicio y el valor práctico del proyecto.

En el sexto capítulo, "*Líneas futuras*", se analizan posibles mejoras y funcionalidades que se podrían añadir al videojuego.

En el séptimo capítulo, "*Referencias*", quedan anotadas todas las referencias que se han usado tanto para el desarrollo del proyecto como para redactar esta memoria.

Finalmente se añaden un anexo en el que se encuentran la planificación y presupuesto final y se analiza como ha sido la desviación entre estas y se sacan conclusiones para futuros proyectos.

Capítulo 2:

Estado de la cuestión

En este capítulo se da una visión resumida de la historia de los videojuegos, desde su origen hasta el momento presente. También se verá un pequeño análisis de la evolución del primer sokoban hasta la multitud de versiones que hay en la actualidad. Y finalmente se analizarán algunos ejemplos de motores gráficos disponibles en la actualidad y las razones de la elección de 3D Game Studio para la implementación del proyecto.

2.1 Historia de los videojuegos

Según las diferentes interpretaciones o definiciones de qué es un videojuego se han establecido diferentes fechas para el origen de de estos. Lo que es evidente es que para que se pueda hablar de un videojuego como tal, tiene que haber movimiento de video en la pantalla.

Partiendo de esta definición, no se considera como origen, el juego de lanzamiento de misiles creado por Tomas T. Goldsmith {11} y Estle Ray Mann que patentaron en **1947** y que estaba basados en sistemas de radar de la segunda guerra mundial y funcionaba con válvulas y usaban rayos catódicos, pero los objetivos ya estaban sobreimpresionados en la pantalla con lo que en realidad no había video en movimiento {12}. Tampoco se considera como origen el tres en raya {13} que presentó como tesis doctoral Alexander Sandy en **1952**. En la figura 4 se puede observar un par de capturas de estos predecesores de los videojuegos.

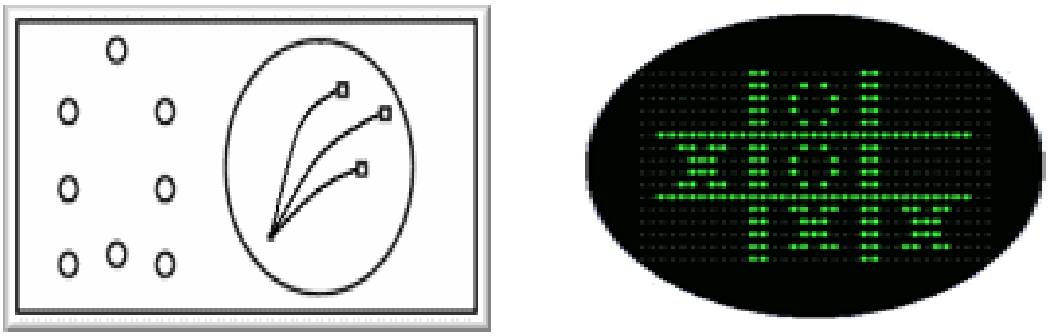

Figura 4: Lanzamiento de misiles (1947) y Tres en raya (1952).

En **1958** aparece el primer videojuego, lo desarrolló William Higginbotham basándose en un programa de cálculo de trayectorias y un osciloscopio. Era un simulador de tenis y se convertía en el primer juego entre dos jugadores. No llegó a patentarlo, lo usó como entretenimiento para los visitantes de una exposición y lo llamó Tennis for Two {14}.

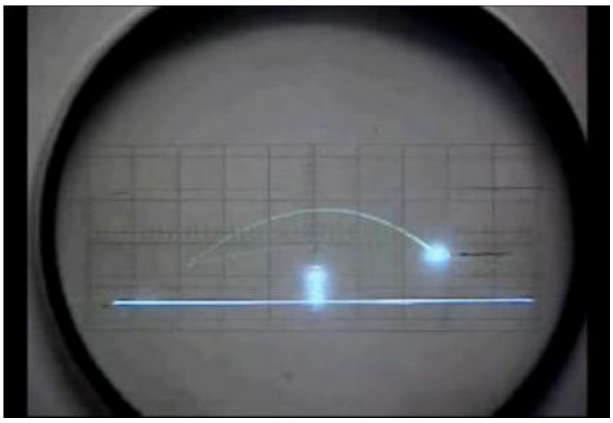

Figura 5: Tennis for two (1958).

Los 50 pasaron como un periodo de intentos frustrados de comienzo de los videojuegos, todos lo que se embarcaban en tal empresa, desistían rápidamente indicando que era una pérdida de tiempo. Hasta que en **1961** Steve Russell creo **Space War** {15} en una máquina PDP-1 (Programmed Data Processor-1) {16}. Consistía en dos naves manejadas por dos jugadores diferentes que intentaban dispararse entre ellas. Esté se convirtió en el primer videojuego de ordenador de la historia. No llegó a tener éxito fuera del ámbito universitario, porque su creador no pretendió su comercialización ya que requería una plataforma hardware valorada en 120.000 $.

Figura 6: PDP-1 (1960).

En **1966** surge la figura de Ralph Baer {17}, al que la historia le acabado reconociendo años y pleitos después como el creador de la primera consola, ya que por mucho tiempo se consideró que fue Nolan Bushnell fundador de Atari. Por aquel año junto con Bill Harrison y Bill Rusch, retomaron un proyecto que consistía en una maquina que conectada directamente a la televisión permitiera jugar al telespectador con dicho aparato. En **1967** terminaron el primer prototipo, llamado "*Brown box*", que ya tenía incorporado unos juegos pregrabados. Pero por segunda vez, Baer no fue capaz de hacer que los ejecutivos de las empresas a las que presentó el proyecto apostaran por este.

Figura 7: Brown box (1967).

Pero Baer no se da por vencido y en Enero de **1968** patenta sus conceptos sobre los videojuegos y sigue ofreciendo su "Brown box" a diferentes empresas entre ellas RCA (Radio Corporation of America, General Electric, Zenith Silvania o Magnavox aunque sin éxito. En Julio de ese mismo año un directivo de RCA que pasó a trabajar para Magnavox {18}, al que sí la

había gustado la propuesta de Baer convenció al resto de directivos y se puso en marcha el proyecto. En **1972** se presenta la "Magnavox Odissey" {19}, con un rifle como accesorio adicional y con 10 videojuegos adicionales. Pero no se llega a comercializar hasta 1978, que se llegaron a vender en las navidades de ese año unas 130000 unidades, pero la consola no llegó a triunfar, sobre todo por unos errores de marketing que limitaron mucho el mercado potencial de esta. Entre estos errores destacaba que, según la publicidad sólo se podía utilizar en televisiones de la misma marca, cuando no era cierto. O que sólo se vendían en franquicias de Maganavox.

Figura 8: Magnavox Odissey y su creador Ralph Baer (1978).

Si a Ralph Baer le costó tiempo conseguir que se le reconociera como el creador de la primera videoconsola fue por la figura de Nolan Bushnell {20}. Nolan desde 1968 veía los videojuegos como un negocio del mundo del entretenimiento con mucho futuro. Junto a Ted Baney, ambos ingenieros de la empresa Ampex, construyeron una nueva máquina para la ejecución de un programa, esta máquina no poseía CPU, muy costosa por aquel entonces. Usaban componentes sencillos creados por ellos mismos. En **1971** deciden asociarse, creando la empresa Syzygy Engineering, para presentar su proyecto a Nutting Asociates {21}. Esta aceptó y sacó al mercado Computer Space {22} con un diseño futurista. Pero al final no tuvo gran éxito y finalizaron su contrato con Nutting. Por temas de derechos de autor, Nolan y Baney cambian el nombre de su empresa por la de Atari {23}.

Pero Nolan no se dio por vencido y en **1972** estaba presente entre los asistentes de la demostración de Magnavox Odissey, incluso llegó a jugar al Ping-Pong. Tras esta demostración contrata a otro ingeniero de Ampex, Alan Alcorn, para que haga una versión arcade de este juego al que llamaron **Pong** {24}. Esta era una versión mejorada del juego de Baer, incluía puntuación, sonido y mejor rutina de movimientos. Ofrecieron su nuevo proyecto a Bally

Midway que gestionaba el negocio de las máquinas de pinball y a Nutting Associates, pero ambos lo rechazaron. Probaron la máquina terminada en una taberna y fue tal el éxito de esta, que la máquina se averió porque el depósito de las monedas se había llenado en un día, esto animo a Nolan, a que Atari se encargara de la fabricación y distribución de la máquina. Con las primeras onces máquinas que distribuyo con facilidad, pudo completar un pedido de 150 máquinas más. Posteriormente pidió un préstamo de 50.000$ para poder contratar operarios y ampliar sus infraestructuras. Fruto del éxito de Atari surgieron empresas por otros países que la imitaron, como Taito {25} en Japón, y ya en **1973** habían surgido 15 empresas dedicadas al negocio de los videojuegos. Llegó a generar 250 millones de dólares anualmente, era el nacimiento de la industria del videojuego.

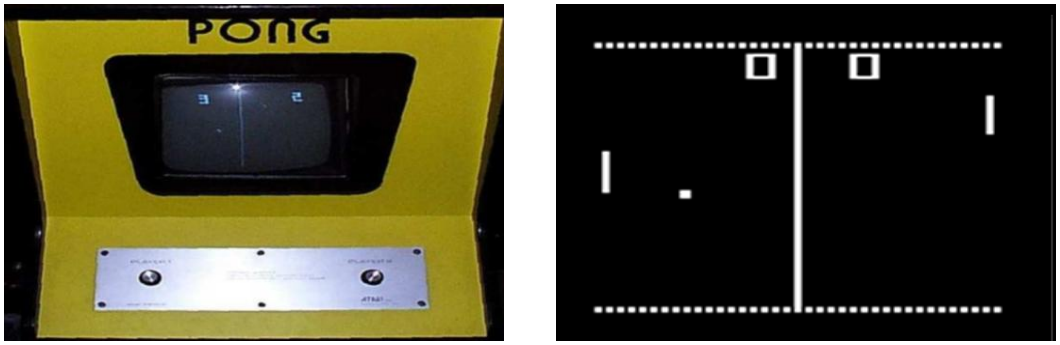

Figura 9: Máquina recreativa Pong de Atari (1972)

Mientras que la competencia de Atari sacaba títulos que eran copias de Pong, esta fruto de la política de empresa liberal, que fomentaba una fuerte motivación entre sus empleados, sacaba multitud de títulos innovadores, como Tank {26} o Space Race {27}.

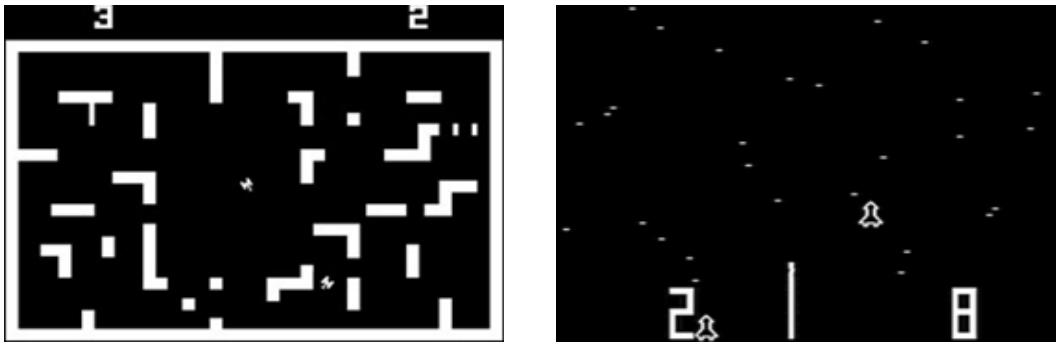

Figura 10: Tank (1974) y Space Race (1973).

En **1975** Atari saca su primera consola para el hogar, con un éxito rotundo, vendiendo en las primeras navidades 150.000 unidades. Y para 1977, ya se habían vendido 13 millones de unidades en total.

Figura 11: Primera consola Atari (1975)

En la década de los 70 llegan los microprocesadores que permitía pasar de los videojuegos que se implementaban con circuitería TTL {28}, que cualquier cambio implicaba cambio en hardware, a modificación en software. Se deja de "*usar*" la figura de ingeniero y se pasa a la de programador.

En **1976** Atari tiene un nuevo proyecto y es sacar la primera consola con un microprocesador, la nueva Atari VCS, pero por falta de presupuesto para un proyecto tan ambicioso tiene que esperar a **1976** en el que la Warner Bross compra la empresa por 28 millones de dólares. Finalmente sale al mercado en Octubre de 1977, con un catálogo de juegos mayor y más innovador que el de la competencia.

Figura 12: Consola Atari VCS (1977)

En **1978** Nolan deja Atari, fruto del alejamiento con la nueva dirección formada por la Warner Bross, y se pone de presidente a Ray Kassar, con muy poca experiencia y que tomó varias decisiones erróneas que hace que la supremacía de Atari desparezca. En este año, Taito

saca un juego con un éxito de proporciones impresionantes, el Space Invaders {29}. El éxito de Atari VCS y Space Invaders, hace que los videojuegos vivan su edad de oro en la década de los 80. Aparecen salones recreativos por doquier. En **1980** aparece Pacman {30} creado por la empresa Namco, que no sólo logro atraer al sector femenino de la población y ser un éxito rotundo sino que fue el primer juego que llevo paralelamente un negocio de Merchandaising.

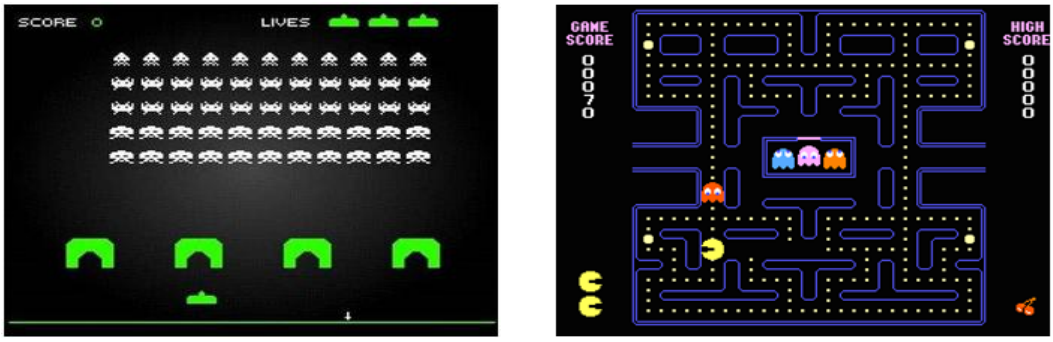

Figura 13: Space Invaders (1978) y Pacman (1980)

En este mismo año, Gumpei Yokoi, jefe de juguetes de Nintendo {31}, observa a un hombre jugueteando con una calculadora y se le ocurre la idea de crear las primeras máquinas de videojuegos portátiles, llamadas "*Game and Watch*" {32}. Era muy parecida a una calculadora que permitía ejecutar acciones sobre la imagen, permitiendo que el personaje cambiara de posición en la pantalla LCD, dando la sensación de movimiento. Tenían muy bajo coste de producción. Cada máquina incorporaba un sólo juego sin posibilidad de jugar a otros. Nintendo llego a sacar unos 59 modelos diferentes, entre los que destacaban Donkey Kong y Mario Bros. En los siguientes once años llegaron a vender once millones de unidades.

Figura 14: Game and Watch Donkey Kong, de Nintendo.

En 1980 también llega **la generarción de los 8 bits,** apareciendo ordenadores de sobremesa. Entre estos destacaron Zx Spectrum (1982) {33}, que destacaba por su optimizado y compacto diseño. En este mismo año sale al mercado Comodore 64 (1982) {34}, que

incorporaba además de una unidad de casete una disquetera de 5 1/4, fue junto con Spectrum, de los microprocesadores más importantes de su época, de hecho a día de hoy tienen una comunidad de usuarios que siguen programando aplicaciones. Dos años después, surge Amstrad CPC (1984) {35}, para competir contra sus dos rivales directo, anteriormente mencionados. Este además de incorporar unidad de casete y disquetera, permitía el uso del monitor como pantalla de televisión, gracias a un adaptador de TV. El surgimiento de estos, hizo que el sector de los videojuegos entrara en crisis ya que hizo que las ventas disminuyeran drásticamente.

Figura 15: Zx Spectrum, Comodore 64 y Amstrad CPC

El surgimiento de estos microprocesadores provocan un efecto devastador en el mundo de las consolas ya que hace que disminuyan radicalmente el número de ventas, ya que a estas últimas se las veía solo como elementos de ocio y entretenimiento, mientras que a los microprocesadores se les veía como algo mucho más útil, que podía tener un valor académico a la hora ayudar a los estudiantes en sus estudios.

En **1983** Nintendo, de la mano de su presidente Hiroshi Yamahuchi {36} y el creador de juegos Shigeru Miyamoto {37}, irrumpe en el mercado Japonés de los videojuegos con su consola "*Famicom*", que posteriormente en **1985** entraría en el mercado europeo y americano con el nombre de **NES** (Nintendo Entertainment System) {38} y {39}. Fue un éxito rotundo, después de la crisis que había pasado el sector. Era una consola de 8 bits, pionera en aspectos destacables de un juego como el diseño de estos y el planteamiento de los mandos. Tenía un sistema de juegos basados en cartuchos, en que cada uno de estos era un juego diferente. Disponía de un amplio catálogo de juegos, creando sagas famosas que llegan a la actualidad como el Super Mario Bross {40}. Era un juego de plataformas de desplazamiento lateral que narraba las aventuras de un fontanero llamado Mario, que ya había salido en otros juegos de Nintendo, que debe rescatar a la princesa del Reino Champiñón, a la que había secuestrado el malvado Browser. Fue un gran éxito de ventas, llegando a venderse más diez millones de cartuchos. A raíz de este título, Mario Bros se convirtió en el icono de la compañía En **1987**, NES fue el juguete más vendido en EEUU, vendiendo millones de consolas y juegos.

Figura 16: NES (1985) y Super Mario Bros

En **1985** Sega {41} saca su consola Master System {42}, basada en la consola NES pero con inferiores prestaciones, que llega a funcionar en ventas mejor en Europa por su estrategia de marketing. Fuera de esta no consigue batir a Nintendo, entre otras razones por el retraso de lanzamiento y el catálogo de juegos disponibles. El icono no oficial de Sega de estos primeros años, es Alex Kidd {43}, el juego más vendido de esta plataforma. Era un juego de plataformas, aunque tenía partes de rol y estrategia.

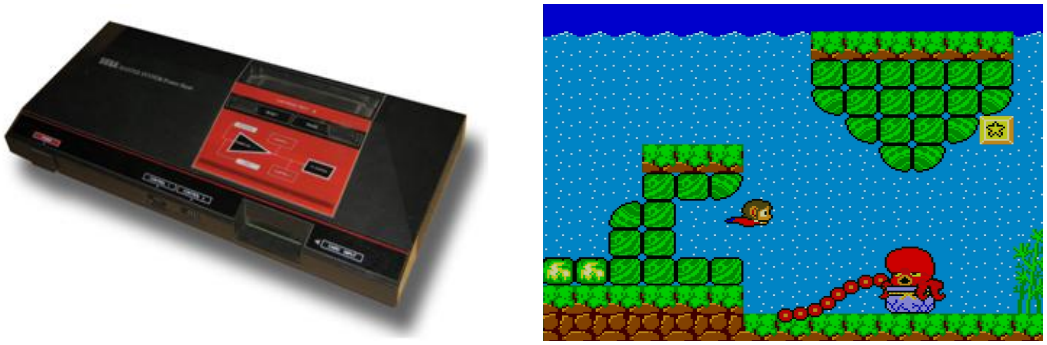

Figura 17: Master System (1985) y Alex Kidd.

En los años siguientes hubo una "*batalla*" por la hegemonía del mercado de las consolas entre Nintendo y Sega. En **1988** Sega saca Mega Drive {44} para competir con NES a la que, salvo en algunos países de Europa y Brasil, no había logrado arrebatarla el liderato en ventas. Se convierte en la primera consola de 16 bits, lo que la daba más calidad gráfica y mayor velocidad de juego. Esto unido a un catálogo de juegos deportivos muy amplio y la creación del que sería su icono oficial, el conocido erizo azul llamado Sonic {45} que era capaz de correr a la velocidad del sonido, hace que Sega se convierta en líder en el mercado de las consolas durante los tres años siguientes. Sonic, era un juego de plataformas con un ritmo de juego

trepidante, en el que jugador tenía que ir superando los enemigos y obstáculos recogiendo el mayor número de anillos.

Figura 18: Mega Drive (1988) y Sonic

En **1989**, Nintendo saca la primera consola portátil de cartuchos intercambiables sucesora de las Games and Watch, Game Boy {46}. A pesar de ser un sistema en blanco y negro (cuatro tonos de gris), como no tenía ningún tipo de competencia, unido al fenómeno Tetris {47} y al esfuerzo por parte de toda la compañía hicieron que las ventas se dispararan. Entre la primera versión y una posterior que sacaron en color, llegaron a vender más de 118 millones de unidades. Tetris es un juego tipo puzle, que consiste en ir colocando las distintas figuras geométricas que van bajando por la pantalla, de tal manera que vayan completando líneas horizontales. Cuando una línea horizontal se completa, se borra de la pantalla. La velocidad de caída de las piezas va aumentando con la dificultad.

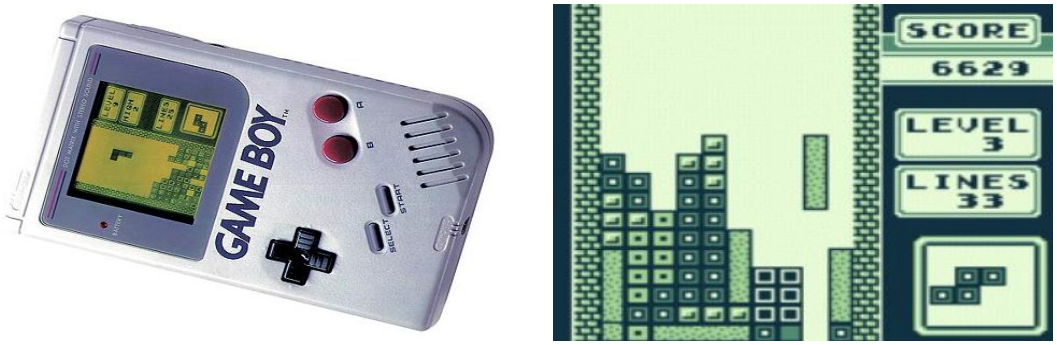

Figura 19: Game Boy (1989) y Tetris

En **1991** Nintendo vuelve a la ataque para intentar recuperar la supremacía perdida a favor de Sega, para ello saca su nueva consola, Supernintendo (SNES) {48}. Tenía un diseño inusual para la época, aunque es el que se sigue en la actualidad, ya que tenía un procesador muy poco potente pero incorporaba potentes chips para el control del video y del sonido. Entre los accesorios que tenía, destacaban su adaptador de Game Boy que permitían jugar a juegos de

la portátil en la SNES, también tenían un multitap lo que le permitía conectar varios mandos a la vez. Tenía un catálogo de juegos disponibles muy alto y con auténticos superventas como el el Super Mario Kart {49}. Este consistía en un juego de carreras con diferentes personajes, con diferentes habilidades y que se podían tirar objetos entre sí. Todo esto hizo que Nintendo volviera a ser líder del mercado, de hecho la Supernintendo llegó a ser la consola de 16 bits más vendida.

Figura 20: Super Nintendo y Super Mario Kart.

Sega intento contrarrestar sacando varias consolas como Sega Saturn {50} y Sega CD, que se convirtieron en las primeras en pasar del uso de cartuchos a CDs. Pero tuvieron un éxito relativo en algunos países como Japón, con juegos como el Virtua Fighter {51}, uno de los primeros en utilizar tecnología 3D. Nintendo por su parte, sacó Nintendo 64 {52} que incorporaba una tecnología de 64 bits, pero seguía usando cartuchos que suponía una seria desventaja frente a sus competidores que habían pasado a la tecnología CD, mucho más novedosa y más barata.

Estos intentos tanto de Sega como de Nintendo fueron infructuosos y entre 1994 y 1995 la venta de consolas baja en casi un 20%.

Figura 21: Sega Saturn (1994) y Nintendo 64 (1996)

En estos mismos años, Sony {53} y Nintendo tenían un acuerdo por el cual, Sony desarrollaría el CD-ROM para la nueva SNES PlayStation, pero Nintendo al ver el fracaso de Sega CD decidió no seguir adelante, pero Sony decide seguir por su cuenta y en diciembre de **1994**, en Japón y en Septiembre de **1995** en EEUU, se mete el mundo de los videojuegos a lo grande, sacando a la venta una consola propia, PlayStation {54}. Utilizaba soporte CD-ROM para sus juegos, con una calidad gráfica y un catálogo de juegos disponible hizo que no tuviera competencia. Sólo en EEUU vendiendo 100.000 unidades en su primera semana. En seis meses las ventas ascendían a un millón, y dos años después ya se habían vendido en todo el mundo 20 millones de unidades. Estableciéndose como líder indiscutible del mercado muy lejos de sus competidores Nintendo 64 y Sega Saturn. Entre los juegos más destacados se encontraba el primero de una saga que continua a día de hoy, Resident Evil 1 {55}. Un juego de acción y terror, en el que el jugador controlaba a un policía en una mansión plagada de zombis. Un juego novedoso en su género que llegó a vender millones de copias.

Figura 22: PlayStation (1994) y Resident Evil 1 (1996)

Nuevamente en el **2000** Sony vuelve a dar un golpe sobre la mesa, adelantándose a sus competidores, sacando PlayStation 2 {56} un año antes que el resto, lo que la permitió convertirse en la consola más vendida de toda la historia. Era una consola de 128 bits con lector de DVD incorporado lo que hacía más atractiva su compra. Se distinguía por tener un potente procesador central, Emotion Engine {57}, y sus mandos ergonómicos y con vibración. Además permite jugar online en un amplio catálogo de juegos. Es la única consola que no sólo ha competido con sus contemporáneas sino que ha llegado a competir con la siguiente generación de consolas, estando más de once años en activo y llegando a vender más de 150 millones de unidades. En cuanto a juegos superventas de esta consola se podrían comentar muchos, a destacar la continuación de la saga de Resident Evil con su cuarta entrega {58}, o juegos de futbol como FIFA {59} y ProEvolution Soccer {60}, que año tras año lograron vender millones de copias.

Figura 23: PlayStation 2 (2000) y Resident Evil 4 (2005)

Un año después en **2001** Microsoft {61} hace su entrada en el mercado de las consolas con Xbox {62}, una máquina potente como la PlayStation 2 y aunque tuvo buenas ventas no pudo igualar a Sony que ya había copado el mercado. La principal característica de esta consola era su procesador basado en el procesador Intel Pentium III {63}. Como su arquitectura está basada en la de un PC, facilitó mucho a los desarrolladores de juegos adaptar títulos de PC a Xbox. Entre los juegos más destacados esta Halo {64}, un juego de acción en primera persona.

Figura 24: Xbox (2001) y Halo (2001)

Por su parte Nintendo saco Game Cube que no lo logró atraer al sector adulto y fue un fracaso comercial, lo que hizo a la empresa nipona centrar todos sus esfuerzos en la diferentes versiones de Game boy.

Finalmente en el **2005** aparece la última generación de consolas. Microsoft aprende de la lección que le dio Sony en el 2000 y se adelanta sacando primero la Xbox360 {65}. Entre sus principales características es que sigue con soporte DVD y que tiene integrado Xbox live, servicio online a través del que todos los usuarios pueden jugar en red, descargar juegos, música y películas. Recientemente ha sacado un novedoso periférico llamado Kinect {66} que permite ser al jugado el propi mando. Es una cámara con sensores que recoge el movimiento

del jugador y mediante este controla la máquina. Sony no se hizo esperar y unos pocos meses después, en el **2006**, saca PlayStation 3 {67} con la incorporación de la tecnología Blu-Ray, pero no llega a tener el éxito previsto, entre otros motivos por su alto coste inicial. Entre el gran catálogos de juegos que tiene a aparece una versión más de una saga que ha estado presente en todas las consolas de Sony, Metal Gear Solid 4 {68}.

Figura 25: Xbox360 (2005), PlayStation 3 (2006) y Kinect (2011)

La última consola en llegar es la Wii {69} en **2006** con la que reaparece en el mercado Nintendo, un concepto innovador de su sistema de control por movimiento y juegos sencillos para toda la familia, como Wii Sports {70} hace que supere en ventas a sus competidoras.

Figura 26: Wii de Nintendo (2006) y Wii Sports

Si algo ha demostrado la industria del videojuego es que está en constante evolución, por eso mismo las diferentes compañías ya han anunciado que están trabajando en la siguiente generación de consolas, de hecho Nintendo sacará Wii 2 al mercado en **2012.** A juzgar por los números de ventas que muestran la tabla 1 sacada de la web de Abadía Digital {71} de todas las consolas a lo largo de la historia, parece que seguro que volverá a ser un gran negocio.

Puesto	Consola	Unidades Vendidas
1	PlayStation 2	150 millones
2	Nintendo DS	144,59 millones
3	Game Boy	118,69 millones
4	PlayStation	102,49 millones
5	Wii	84,64 millones
6	Game Boy Advance	81,51 millones
7	PSP	67,8 millones
8	NES	61,91 millones
9	Xbox 360	50 millones
10	Super Nintendo	49,1 millones
11	PlayStation 3	47,9 millones
12	Nintendo 64	32,93 millones
13	Atari 2600	30 millones
14	MegaDrive	29 millones
15	Xbox	24 millones
16	GameCube	21,74 millones
17	Master System	13,4 millones
18	GameGear	11 millones
19	DreamCast	10,6 millones
20	TurboGrafx	10 millones
21	Saturn	10 millones
22	Colecovision	6 millones
23	Intellivision	3 millones
24	Magnavox Odyssey²	2 millones
25	Atari Jaguar	250.000

Tabla 1: Número de ventas por consola

2.2 Historia del sokoban

Sokoban {72} es un juego tipo puzle o rompecabezas que se creó en Japón en 1980 por Hiroyuki Imabayashi. Salió a la venta por primera vez de la mano de Thinking Rabbit, una compañía de juegos japonesa. La primera versión que salió al mercado europeo fue publicada por la empresa Spectrum Holobyte en 1984.

El nombre del juego significa encargado de almacén consiste en que el jugador controla un personaje que tiene como objetivo colocar en las posiciones marcadas una serie de cajas que se encuentran distribuidas a lo largo de todo el nivel. La dificultad se encuentra en que los

movimientos son limitados porque solo puede empujar las cajas y no tirar de ellas, con lo que si coloca una caja en una esquina no podrá sacarla de ahí.

La sencillez de la dinámica del juego ha hecho que hayan salido multitud de versiones de este y que continuamente se creen nuevos niveles que van dificultades sencillas hasta verdaderos retos mentales que pueden llevar días resolverlos. Entre las versiones oficiales destacan Boxxle {73} y Sokoban DS {74}. Ambas versiones salieron para la portátil de Nintendo, el primero para la primera Game Boy. El segundo, más de veinte años después la Nintendo DS. En la figura 27 se puede ver claramente la evolución gráfica en este juego, aunque como se observa la dinámica de este no ha variado en nada.

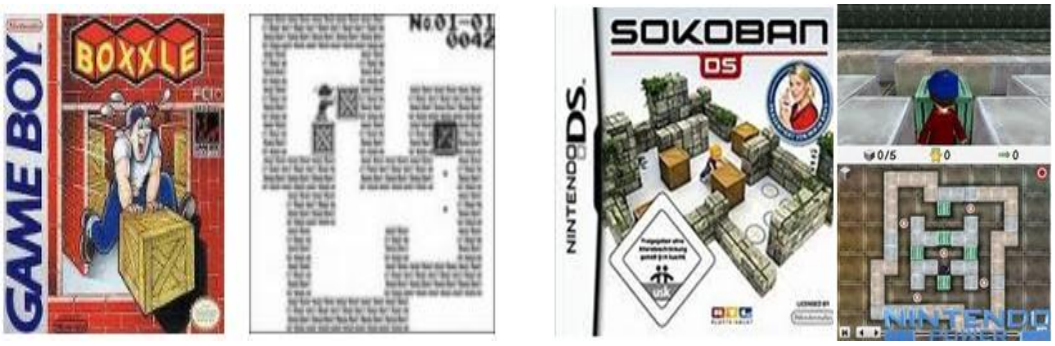

Figura 27: Boxxle (1989) y Sokoban DS (2010)

En internet se pueden encontrar cientos de versiones y niveles basados en el Sokoban. Solo en la página de la referencia {75}, tiene más de sesenta enlaces diferentes a páginas que tiene otra versión de sokoban disponible. Como por ejemplo, Magic Crates {76}, en el que controlas a una chica que se ha perdido en el bosque y tiene que ir colocando las cajas para poder seguir avanzando de nivel para poder salir de este.

Figura 28: Magic Crates

2.3 Otros motores gráficos y 3D Game Studio A7

Se entiende como motor gráfico al conjunto de programas que entrelazados permiten la creación, entre otras cosas, de un videojuego. Desde que los videojuegos entran en el mundo 3D, empiezan a surgir estos motores que permiten hacer este efecto 3D utilizando figuras, texturas y planos superpuestos que dan sensación de profundidad. En la actualidad la mayoría de los motores utilizan rutinas preestablecidas para controlar la tarjeta gráfica como son el caso de Direct3D {77} para Windows.

La revista digital Neoteo de ABC {78}, hizo una clasificación de los 10 motores gráficos más destacados de los que se van a destacar los tres primeros:

Unreal Engine 3 {79}**,** Epic Games {80} es la propietaria de este motor y la primera versión se creó inicialmente para la implementación de Unreal en 1998, un juego en primera persona de acción. Posteriormente se ha convertido en el motor más usado para el desarrollo de videojuegos por las compañías más importantes. Esta tercera versión surgió en el 2006. En 2007 se asociaron con Sony para adaptar este motor al uso de la PlayStation 3. Con este se han creado juegos como Gears of war {81} o Bioshock. Es el que mejor se adapta a todas las plataformas y facilita la jugabilidad online. Además en 2009, Epic Games liberó la aplicación permitiendo su uso gratuito y sólo facturan un porcentaje de las ventas en caso de comercialización del producto desarrollado. También tiene usos en sectores no relacionados con los videojuegos como simulaciones de construcción, simuladores de conducción y generación de terrenos.

Figura 29: Gear of war y Bioshock

CryEngine 2 {82}**,** es el más avanzado y no necesita de ningún complemento para producir sus sonidos, animaciones o físicas. Este año se ha presentado la tercera versión de este. Entre los juegos desarrollados sobre este motor resalta Crysis 2 {83}.

Anvil Engine, creado en 2007 por Ubisoft {84}, aunque no es tan conocido como los dos primeros hay que decir que está detrás de juegos de los denominados "superventas", como Assassin's Creed {85} y Prince of Persia. Es capaz de crear animaciones a tiempo real en escenarios abiertos y totalmente escalables.

Figura 30: Crysis 2 (CryEngine 2) y Assassin's Creed (Anvil Engine)

Aunque esta revista no mencionaba el motor gráfico **XNA** {86} de Microsoft, es uno que se ha tenido en cuenta como candidato para la realización del proyecto. Entre otros motivos porque es gratuito y permite el desarrollo de aplicaciones para PC y Xbox 360. Además tiene una gran comunidad de desarrolladores y hay multitud de información disponible en la red.

Finalmente para la realización del proyecto se ha decidido usar 3D Game Studio A7 por las siguientes razones:

- Dispone de una versión totalmente gratuita y libre.
- Además del motor gráfico, trae varias aplicaciones gratuitas para la creación de los personajes y niveles, además de un editor de código.
- Se pueden llegar a implementar sencillo juegos sin llegar a escribir ni una sola línea de código.
- El lenguaje de programación es Lite-C, que es una versión simplificada de C++, lo que facilitará mucho el aprendizaje y dominio de este.
- El cliente tenía experiencia en proyectos anteriores con versiones más antiguas de este motor, con lo que podría aportar dicha experiencia en las diferentes fases del proyecto.
- En el caso de querer llegar a comercializar finalmente la aplicación no sería muy costosa adquirir la liciencia para su distribución.

Capítulo 3:

Análisis, diseño, planificación e implementación

En este capítulo se verán todas las fases de la elaboración del proyecto, para ello está divido en tres partes:

En la primera parte se hará un fase de análisis, en la que se hace una toma de requisitos de la aplicación, los casos de uso y los diagramas de flujo de cómo va a funcionar dicha aplicación.

En la segunda parte, se hará un diseño de cómo tiene que ser la aplicación a nivel de detalles del juego, tales como personajes, escenarios e interfaz con el usuario.

Y finalmente, en la última parte se lleva a cabo la implementación del proyecto, dónde se puede ver el código y los pasos para la creación de los escenarios y la interfaz gráfica con el usuario.

3.1 Análisis

Esta primera fase es de vital importancia ya que un análisis deficiente, podría tener como consecuencia que el resultado final no se ajustara en absoluto a lo que quiere el cliente y habría que rehacer el proyecto entero. Por eso primeramente se indica cuál es la idea inicial del juego, la dinámica de este y cómo se ha planteado la jugabilidad y las posibilidades que ofrece al jugador.

Posteriormente se presenta una toma de requisitos en la que se mantienen varias entrevistas con el cliente, para poder anotar que funcionalidades quiere que tenga la aplicación. También se hacen los casos de usos de esta, documentando y haciendo diagramas de flujo indicando la relación del sistema con el usuario y otras aplicaciones.

Todo esto servirá para la siguiente fase de desarrollo, en la que habrá que plasmar toda esta información recogida, en bocetos de lo que será la realidad, de ahí la importancia de esta primera fase que se comentaba anteriormente.

3.1.1 Idea inicial

Como ya se vio en el primer capítulo, la idea inicial es hacer un juego en tres dimensiones. Esté consistirá en diferentes niveles y en cada uno habrá un escenario, con una serie de cajas distribuidas por este y una zona donde deben ser colocadas. Además habrá un personaje que es el que manejará el jugador, que se podrá mover por el escenario y empujar las cajas para llevarlas a la zona designada.

En cuanto a la cámara en el juego, se ha planteado que haya varias perspectivas, una en tercera persona, una desde arriba que "*vea*" todo el nivel y otra desde arriba pero más cercana que siga al personaje en sus movimientos por el nivel.

También se quiere que por lo menos haya un par de personajes diferentes entre los que pueda elegir el jugador.

Otras características importantes son que el juego tenga música y sonido, que se puedan guardar y cargar partidas y que se pueda elegir entre diferentes niveles de dificultad.

Por supuesto, se desea que el juego tenga una serie de menús que al jugador le permita modificar las opciones configurables, mencionadas anteriormente como el personaje, dificultad y que la música este habilitada o no.

Finalmente se desea que el juego tenga un sistema de puntuación basado en el tiempo que le lleva al jugador superar los diferentes niveles y que posteriormente estos records puedan ser guardados y consultados posteriormente.

3.1.2 Requisitos de Usuario

En este punto se hace la toma de requisitos en sí, esta consiste en poner en "papel" lo que quiere el cliente, sus funcionalidades y características. Para ello se han mantenido varias entrevistas con el cliente para hacer una lista clara y que cubra todos los detalles relevantes de la aplicación.

Se van a definir los requisitos de usuario con este conjunto de características:

1. **Identificación**: Este campo indicará el número de requisito de usuario de capacidad del sistema

2. **Necesidad**: Clasifica la necesidad de este requisito en tres niveles, esencial, deseable y opcional.

3. **Título**: Nombre descriptivo del requisito

4. **Descripción**: Explica en qué consiste este requisito

Identificador:	**RUC-01**	Necesidad:	Deseable
Título:	Fichero ejecutable		
Descripción:	La aplicación debería tener un fichero que permita la ejecución directa de esta, para facilidad del usuario final.		

Tabla 2: RUC-01. Ejecutable.

Identificador:	**RUC-02**	Necesidad:	Esencial
Título:	Controles		
Descripción:	La aplicación deberá permitir que el usuario pueda utilizar el teclado y el ratón para interactuar con esta.		

Tabla 3: RUC-02. Controles.

Identificador:	**RUC-03**	Necesidad:	Esencial
Título:	Menús		
Descripción:	La aplicación deberá tener un sistema de navegación de pantallas o menús que permita al usuario ir configurando las diferentes opciones que le permita dicha aplicación. Además de iniciar y finalizar el juego.		

Tabla 4: RUC-03. Menús

Identificador:	**RUC-04**	Necesidad:	Deseable
Título:	Configuraciones		
Descripción:	La aplicación deberá guardar la configuración elegida mientras se está jugando la partida.		

Tabla 5: RUC-04. Configuraciones

Identificador:	**RUC-05**	Necesidad:	Esencial
Título:	Guardar partidas		
Descripción:	La aplicación deberá permitir guardar partidas con los puntos que lleva el jugador, el nivel en el que está y las configuraciones elegidas, tales como dificultad y personaje elegido.		

Tabla 6: RUC-05. Guardar partidas

Identificador:	**RUC-06**	Necesidad:	Deseable
Título:	Cargar partidas		
Descripción:	La aplicación deberá permitir cargar partidas guardadas previamente.		

Tabla 7: RUC-06. Cargar partidas.

Identificador:	**RUC-07**	Necesidad:	Deseable
Título:	Elegir personaje		
Descripción:	La aplicación debería permitir al jugador elegir entre al menos dos personajes diferentes para jugar la partida.		

Tabla 8: RUC-07. Fichero ejecutable.

Identificador:	**RUC-08**	Necesidad:	Esencial
Título:	Control personaje		
Descripción:	La aplicación deberá permitir al usuario mover al personaje de arriba a abajo y de izquierda a derecha del nivel, mediante las teclas cursoras del teclado.		

Tabla 9: RUC-08. Control personaje.

Identificador:	**RUC-09**	Necesidad:	Opcional
Título:	Control personaje 2		
Descripción:	El usuario pueda configurar cuáles serán las teclas del teclado con las que manejara al personaje.		

Tabla 10: RUC-09. Control personaje 2.

Identificador:	**RUC-10**	Necesidad:	Esencial
Título:	Cámara		
Descripción:	La aplicación deberá tener al menos tres vistas diferentes, una en tercera persona y dos que vean el nivel desde arriba, una de estas que capte todo el nivel y otra más cercana que se mueva con el movimiento del personaje.		

Tabla 11: RUC-10. Cámara.

Identificador:	**RUC-11**	Necesidad:	Esencial
Título:	Posición cámara		
Descripción:	La aplicación deberá permitir al usuario cambiar en cualquier momento la vista entre las tres disponibles que hay.		

Tabla 12: RUC-11. Posición cámara.

Identificador:	**RUC-12**	Necesidad:	Esencial
Título:	Sistema de puntuación		
Descripción:	La aplicación debe tener un sistema de puntuación basado en el tiempo que le lleva al jugador superar los diferentes niveles.		

Tabla 13: RUC-12. Sistema de puntuación.

Identificador:	**RUC-13**	Necesidad:	Esencial
Título:	Tiempo		
Descripción:	La aplicación deberá medir el tiempo que le lleva al usuario superar cada nivel.		

Tabla 14: RUC-13. Tiempo.

Identificador:	**RUC-14**	Necesidad:	Esencial
Título:	Sistema operativo		
Descripción:	La aplicación deberá funcionar en cualquier versión de Windows xp o superior.		

Tabla 15: RUC-14. Sistema operativo.

Identificador:	**RUC-15**	Necesidad:	Esencial
Título:	Movimiento de cajas		
Descripción:	Las cajas, que forman parte del juego, deben moverse al ser empujadas por el personaje de arriba abajo y de derecha a izquierda por el nivel. No deben rebotar con las paredes ni se debe poder tirar de ellas.		

Tabla 16: RUC-15. Movimiento de cajas.

Identificador:	**RUC-16**	Necesidad:	Esencial
Título:	Música de fondo		
Descripción:	La aplicación deberá permitir al usuario habilitarla y deshabilitarla.		

Tabla 17: RUC-16. Música de fondo.

Identificador:	**RUC-17**	Necesidad:	Deseable
Título:	Música de fondo 2		
Descripción:	La aplicación debería tener algún sonido para cuando el usuario coloca una caja en su posición correcta.		

Tabla 18: RUC-17. Música de fondo 2.

Identificador:	**RUC-18**	Necesidad:	Deseable
Título:	Música de fondo 3		
Descripción:	La aplicación debería tener músicas diferentes para cuando el usuario esté navegando por los menús del juego y cuando esté jugando.		

Tabla 19: RUC-18. Música de fondo 3.

3.1.3 Casos de uso

Una vez hecha la toma de requisitos se pasa a detallar los casos de uso, para definir el comportamiento que debería tener la aplicación. Para ello primeramente se hará una descripción textual de los pasos que debe seguir el usuario para realizar las acciones o funciones que permitirá el sistema. Posteriormente se mostrará un diagrama de flujo que recogerá todos los casos de uso y sus relaciones entre estos. Finalmente se detallarán todos los atributos de cada caso de uso en las tablas 20 a 32.

El usuario deberá iniciar la aplicación y está debe proporcionar un interfaz de pantallas por las que el usuario se podrá mover con el uso del ratón. También deberá poder configurar las opciones del juego, para ello irá del menú principal a un menú de opciones, en este podrá configurar la música del juego, el personaje con el que jugará la partida, la dificultad de esta y podrá ver cuáles son los controles para mover al personaje. Volviendo al menú principal podrá acceder a otra pantalla en las que tenga acceso a las partidas guardadas previamente y poder cargarlas, de tal manera que en vez de iniciar una nueva partida, continuará una en el nivel y con los puntos que llevaba cuando la guardó. Por supuesto, desde el menú principal también podrá empezar una nueva partida. Una vez que empiece a jugar, el usuario deberá poder controlar al personaje. Si supera un nivel en un buen tiempo podrá establecer un nuevo record

y si lo desea, guardará la partida para continuarla más tarde. También podrá salir al menú principal y si lo desea salir de la aplicación.

A continuación se puede observar en la figura 31 el diagrama de flujo de los casos de uso.

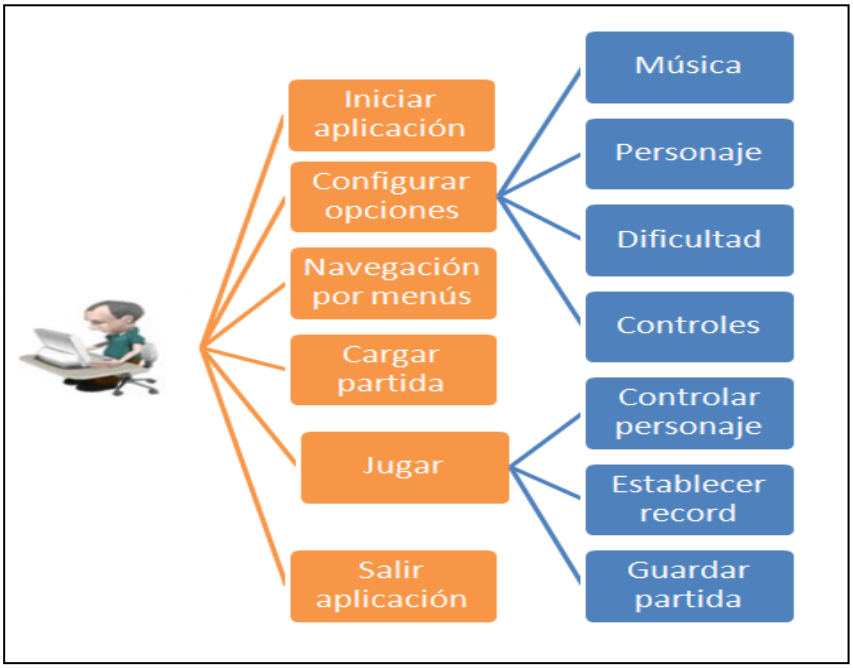

Figura 31: Diagrama de casos de uso.

Finalmente al igual que se hizo con la toma de requisitos, y para hacer una explicación más extensa de cada caso de uso se definen las siguientes características de las tablas en las que se describirán cada uno de estos:

- **Identificador**: Este campo indicará el número de casos de uso.
- **Título**: Nombre descriptivo del caso de uso.
- **Actores**: Quiénes intervienen en cada caso de uso.
- **Objetivo**: Finalidad y objetivo a conseguir con cada caso de uso.
- **Precondiciones**: Condiciones necesarias para que se pueda dar ese caso de uso.
- **Post-condiciones**: Situación siguiente que se da en la aplicación una vez se ha dado el caso de uso.

Identificador:	**CU-01**
Título:	Iniciar aplicación
Actores:	Jugador
Objetivos:	Iniciar la aplicación
Precondiciones:	La aplicación debe estar instalada en el ordenador en el que se vaya a usar y tenga un sistema operativo compatible con esta.
Post-condiciones:	Aparecerá en pantalla el menú principal del juego.

Tabla 20: CU-01. Iniciar aplicación

Identificador:	**CU-02**
Título:	Configurar opciones
Actores:	Jugador
Objetivos:	Permitir al usuario acceder a la configuración de opciones.
Precondiciones:	Tener en pantalla el menú principal
Post-condiciones:	Se cargará el menú de opciones

Tabla 21: CU-02. Configurar opciones.

Identificador:	**CU-03**
Título:	Música
Actores:	Jugador
Objetivos:	Permitir al usuario habilitar o deshabilitar el sonido en el juego.
Precondiciones:	Tener en pantalla el menú de opciones.
Post-condiciones:	En el menú de opciones aparecerá el estado de la música seleccionada, On / Off

Tabla 22: CU-03. Música.

Identificador:	**CU-04**
Título:	Personaje
Actores:	Jugador
Objetivos:	Permitir al usuario elegir entre los personajes disponibles
Precondiciones:	Tener en pantalla el menú de opciones.
Post-condiciones:	En el menú de opciones aparecerá el nombre del personaje seleccionado

Tabla 23: CU-04. Música.

Identificador:	**CU-05**
Título:	Dificultad
Actores:	Jugador
Objetivos:	Permitir al usuario elegir entre los diferentes niveles de dificultad
Precondiciones:	Tener en pantalla el menú de opciones.
Post-condiciones:	En el menú de opciones aparecerá la dificultad seleccionada, fácil, normal o difícil.

Tabla 24: CU-05. Dificultad.

Identificador:	**CU-06**
Título:	Controles
Actores:	Jugador
Objetivos:	Permitir al usuario visualizar cuáles son los controles para poder manejar al personaje.
Precondiciones:	Tener en pantalla el menú de opciones.
Post-condiciones:	Aparecerá en pantalla un recuadro con los controles de manejo del personaje.

Tabla 25: CU-06. Música.

Identificador:	**CU-07**
Título:	Navegación por menús
Actores:	Jugador
Objetivos:	Permitir al usuario navegar por los diferentes menús, y que pueda interactuar con estos a través del ratón.
Precondiciones:	La aplicación este “corriendo” en ese momento
Post-condiciones:	Aparecerá en pantalla el menú solicitado por el usuario

Tabla 26: CU-07. Navegación por menús.

Identificador:	**CU-08**
Título:	Guardar partida
Actores:	Jugador
Objetivos:	Permitir al usuario guardar la partida que está jugando
Precondiciones:	Que esté en el menú de pausa y seleccione la opción de guardar partida.
Post-condiciones:	Aparecerá en pantalla el menú solicitado por el usuario

Tabla 27: CU-08. Navegación por menús.

Identificador:	**CU-09**
Título:	Cargar partida
Actores:	Jugador
Objetivos:	Permitir al usuario cargar una partida previamente guardada
Precondiciones:	En el menú principal haber elegido la opción de cargar partida.
Post-condiciones:	Aparecerá en pantalla el nivel con el personaje y opciones configuradas previamente, cuando se guardó la partida.

Tabla 28: CU-09. Cargar partida.

Identificador:	**CU-10**
Título:	Jugar
Actores:	Jugador
Objetivos:	El uso principal de la aplicación, donde el usuario maneja al personaje para colocar todas las cajas que hay en el nivel.
Precondiciones:	Haber seleccionado el menú principal la opción de “Nueva Partida” o haber cargado una partida previamente guardada.
Post-condiciones:	Aparecerá en la pantalla el nivel con el personaje seleccionado.

Tabla 29: CU-10. Jugar.

Identificador:	**CU-11**
Título:	Controlar personaje
Actores:	Jugador
Objetivos:	Durante el juego, el usuario controla al personaje con el teclado además de poder cambiar la vista de la cámara.
Precondiciones:	Nivel y personajes cargados y visualizados en pantalla.
Post-condiciones:	El usuario controlará al personaje.

Tabla 30: CU-11. Controlar personaje.

Identificador:	**CU-12**
Título:	Establecer record
Actores:	Jugador
Objetivos:	Cuando el jugador supera un nivel, la aplicación le asigna una puntuación en función del tiempo que le ha llevado. Si es una puntuación de record podrá guardar dicha puntuación.
Precondiciones:	Superar un nivel.
Post-condiciones:	Una vez introducido el nombre continúa la partida.

Tabla 31: CU-12. Establecer record.

Identificador:	**CU-13**
Título:	Salir aplicación.
Actores:	Jugador
Objetivos:	Cerrar la aplicación.
Precondiciones:	En pantalla este el menú principal
Post-condiciones:	Se cierra totalmente la aplicación.

Tabla 32: CU-13.Salir aplicación.

3.1.4 Requisitos del Software

A continuación se pasa a analizar los requisitos que deberá tener el futuro software. En concreto se analizarán los requisitos funcionales, los requisitos de rendimiento y por último los requisitos de recursos. Los requisitos funcionales son los que el usuario necesita que efectúe la aplicación y suelen obtenerse, precisamente de los requisitos de usuario de capacidad. En cuanto a los requisitos de rendimiento, se entienden que son aquellos que especifican valores del rendimiento del sistema tales como velocidad de procesado y ejecución. Finalmente los requisitos de recursos, describen los elementos necesarios para poder desarrollar la aplicación y posteriormente ponerla en producción.

Siguiendo con la metodología de los anteriores puntos, para cada requisito se han definido las siguientes características:

- **Identificador**: Este campo indicará el tipo y número de requisito. De tal manera que RSF será requisito funcional, RSR será requisito de rendimiento y por último, RSR será requisito de recursos.
- **Título**: Nombre descriptivo del requisito.
- **Fuente**: Se indica cuál es el origen de este requisito, si es por un requisito de usuario un caso de uso o porque así lo ha estimado el analista.
- **Necesidad**: Clasifica la necesidad de este requisito en tres niveles, esencial, deseable y opcional.
- **Descripción**: Explica en qué consiste este requisito.

Se procede a enumerar los requisitos funcionales:

Identificador:	**RSF-01**	Necesidad:	Deseable
Título:	Auto Set Up		
Fuente:	Analista		
Descripción:	La aplicación final debería estar empaquetada en un archivo final que permita al usuario final instalar de manera sencilla e intuitiva la aplicación.		

Tabla 33: C RSF-01. Auto set up.

Identificador:	**RSF-02**	Necesidad:	Esencial
Título:	Iniciar aplicación		
Fuente:	CU-01 y RUC-01		
Descripción:	Debe tener un fichero ejecutable para iniciar la aplicación.		

Tabla 34: RSF-02. Iniciar aplicación

Identificador:	**RSF-03**	Necesidad:	Esencial
Título:	Mostrar menú de inicio		
Fuente:	CU-02, CU-09 y RUC-03		
Descripción:	Al inicio deberá aparecer el menú principal con las opciones “Nueva partida”, “Cargar partida”, “Opciones” y “Salir del juego”		

Tabla 35: RSF-03. Mostrar menú de inicio.

Identificador:	**RSF-04**	Necesidad:	Esencial
Título:	Mostrar opciones		
Fuente:	CU-02, CU-03, CU-04, CU-05, CU-06 y RUC-02 y RUC-07		
Descripción:	Muestra el menú de opciones configurables por el usuario.		

Tabla 36: RSF-04. Mostar opciones.

Identificador:	**RSF-05**	Necesidad:	Esencial
Título:	Música.		
Fuente:	CU-03, RUC-16 y RUC-17.		
Descripción:	En el menú de opciones se habilita la música y sonidos de la aplicación.		

Tabla 37: RSF-05. Música.

Identificador:	**RSF-06**	Necesidad:	Esencial
Título:	Elegir personaje		
Fuente:	CU-04 y RUC-7.		
Descripción:	En el menú de opciones se deberá poder elegir entre diferentes personajes.		

Tabla 38: RSF-06. Elegir personaje

Identificador:	**RSF-07**	Necesidad:	Esencial
Título:	Elegir dificultad		
Fuente:	CU-05		
Descripción:	En el menú de opciones se deberá poder elegir entre diferentes niveles de dificultad, fácil, normal y difícil. La dificultad variará el tiempo asignado para superar cada nivel.		

Tabla 39: RSF-07. Elegir dificultad

Identificador:	**RSF-08**	Necesidad:	Deseable
Título:	Configurar controles		
Fuente:	CU-11, RUC-8 y RUC-9.		
Descripción:	Es deseable que el usuario pueda configurar los controles para manejar al personaje.		

Tabla 40: RSF-08. Visualizar controles

Identificador:	**RSF-09**	Necesidad:	Esencial
Título:	Visualizar controles		
Fuente:	CU-11, RUC-8 y RUC-9.		
Descripción:	En el menú de opciones se deberá poder visualizar los controles.		

Tabla 41: RSF-09. Visualizar controles

Identificador:	**RSF-10**	Necesidad:	Esencial
Título:	Visualización de juego		
Fuente:	CU-09, CU-10, RUC-12 y RUC-13.		
Descripción:	Mientras se desarrolla el juego se debe ver en pantalla el nivel, el personaje, las cajas. Además de un marcador con el tiempo que queda, el número de cajas colocadas y los puntos conseguidos.		

Tabla 42: RSF-10. Visualización de juego

Identificador:	**RSF-11**	Necesidad:	Esencial
Título:	Control del jugador		
Fuente:	CU-10, CU-11, RUC-02 y RUC-08.		
Descripción:	El usuario contrala al personaje con el teclado. Este se va a poder mover en cuatro direcciones, arriba, abajo, izquierda y derecha del escenario. Podrá girar sobre sí mismo. No podrá saltar ningún obstáculo, lo único que hará es empujar las cajas en la dirección en la que se esté desplazando el personaje, nunca tirar de ellas.		

Tabla 43: RSF-11. Control del jugador

Identificador:	**RSF-12**	Necesidad:	Esencial
Título:	Movimientos de cajas		
Fuente:	Visualización de juego		
Descripción:	Cuando el personaje toca una caja, esta tiene que detectar que la han tocado y moverse en la dirección que la están empujando hasta que dejen de hacerlo o choque contra una pared.		

Tabla 44: RSF-11. Movimiento de cajas.

Identificador:	**RSF-13**	Necesidad:	Esencial
Título:	Posición de las cajas		
Fuente:	CU-10 y RUC-15.		
Descripción:	La aplicación debe verificar cada vez que se mueve una caja si se ha dejado la caja colocada dentro de la zona destinada a tal efecto. De ser así, verificar también si se han colocado el resto de cajas para superar dicho nivel.		

Tabla 45: RSF-13. Posición de las cajas.

Identificador:	**RSF-14**	Necesidad:	Esencial
Título:	Nivel superado		
Fuente:	CU-10, CU-12, RUC-12 y RUC-13.		
Descripción:	Una vez colocada todas las cajas del nivel, la aplicación debe calcular la puntuación obtenida y verificar si está entre los cinco records de mayor puntuación. Si no es un record, mostrará pantalla con nivel completado y opción de continuar si no es ya el último nivel, si es el último nivel sacara pantalla de juego finalizado y la opción de salir. Si es record, mostrará la pantalla con el record conseguido.		

Tabla 46: RSF-14. Nivel superado.

Identificador:	**RSF-15**	Necesidad:	Esencial
Título:	Establecer record		
Fuente:	CU-12 y RUC-12.		
Descripción:	Si se ha conseguido una puntuación de record al superar un nivel, aparecerá una pantalla con la posición de dicho record, la puntuación obtenida y el usuario tendrá que escribir su nombre. Este record se guardará para consultar los records posteriormente.		

Tabla 47: RSF-15. Establecer record.

Identificador:	**RSF-16**	Necesidad:	Esencial
Título:	Cámara		
Fuente:	RUC-10 y RUC-11.		
Descripción:	Si el usuario quiere cambiar de vista bastará con que pulse una tecla del teclado, previamente definida, para cambiar entre las tres disponibles.		

Tabla 48: RSF-16. Cámara.

Identificador:	**RSF-17**	Necesidad:	Esencial
Título:	Fin de tiempo		
Fuente:	CU-10 y RUC-15.		
Descripción:	Cuando se acaba el tiempo asignado para superar un nivel deberá aparecer una pantalla indicándolo y dando la opción de reintentar o dejar la partida.		

Tabla 49: RSF-13. Fin de tiempo.

Identificador:	**RSF-18**	Necesidad:	Esencial
Título:	Mostrar menú pausa		
Fuente:	Analista		
Descripción:	Mientras se está jugando la partida pulsando una tecla del teclado, se debe pausar la partida y aparezca el menú de pausa.		

Tabla 50: RSF-18. Mostrar menú de pausa.

Identificador:	**RSF-19**	Necesidad:	Esencial
Título:	Opciones menú pausa		
Fuente:	Analista		
Descripción:	Debe tener las siguientes opciones: • "Salir de la partida", que volvería al menú principal. • "Reiniciar", que reiniciaría el nivel. • "Guardar partida", aparecería pantalla con el menú de guardar. • "Continuar", seguiría con la partida actual.		

Tabla 51: RSF-19. Opciones menú de pausa.

Identificador:	**RSF-20**	Necesidad:	Esencial
Título:	Guardar partida		
Fuente:	CU-08 y RUC-05.		
Descripción:	En el menú de guardar partida habrá un número limitado de posiciones en las que guardar una partida. El usuario pondrá el nombre que quiera y guardará la partida, sobrescribiendo si hubiera una partida ya guardada en dicha posición.		

Tabla 52: RSF-20. Opciones menú de pausa.

Identificador:	**RSF-21**	Necesidad:	Esencial
Título:	Cargar partida		
Fuente:	CU-09 y RUC-06.		
Descripción:	En el menú de cargar partida, al que se accederá desde el menú principal, aparecerán todas las partidas guardadas previamente y con el ratón seleccionará la partida que se quiera cargar.		

Tabla 53: RSF-21. Cargar partida.

Identificador:	**RSF-22**	Necesidad:	Esencial
Título:	Ver records		
Fuente:	RUC-11, CU-07 y CU-10		
Descripción:	En la pantalla de records, a la que se accederá desde el menú principal, se mostrarán las cinco mejores puntuaciones		

Tabla 54: RSF-22. Ver records.

Identificador:	**RSF-23**	Necesidad:	Esencial
Título:	Desinstalar aplicación		
Fuente:	Analista		
Descripción:	La aplicación podría proporcionar alguna herramienta para una desinstalación sencilla e intuitiva para el usuario.		

Tabla 55: RSF-23. Ver records.

A continuación se enumeran los requisitos de rendimiento:

Identificador:	**RSR-01**	Necesidad:	Deseable
Título:	Carga de la aplicación		
Fuente:	Analista		
Descripción:	La aplicación no debe tardar mucho en cargar el menú principal.		

Tabla 56: RSR-01. Cargar de la aplicación.

Identificador:	**RSR-02**	Necesidad:	Deseable
Título:	Carga de niveles		
Fuente:	Analista		
Descripción:	La aplicación no debe tardar mucho en cargar cada nivel, con todos los componentes de estos y el personaje.		

Tabla 57: RSR-02. Carga de niveles.

Finalmente se enumeran los requisitos de recursos:

Identificador:	**RSRec-01**	Necesidad:	Deseable
Título:	Entorno de desarrollo		
Fuente:	Analista		
Descripción:	Es deseable un PC medianamente potente porque las aplicaciones 3D necesitan de mucha memoria RAM y de un buen procesador para que se pueda llegar a trabajar en un entorno correcto y productivo. Un entorno con unas prestaciones mínimas como estas: • Sistema Operativo: Windows XP o Windows 7 • Memoria RAM: 4 GB • Capacidad: 10 GB de disco duro • Tarjeta gráfica: 512 MB		

Tabla 58: RSRec-01. Entorno de desarrollo.

Identificador:	**RSRec-01**	Necesidad:	Deseable
Título:	Entorno de producción		
Fuente:	Analista		
Descripción:	Para el uso de la aplicación es recomendable tener un PC con las siguiente prestaciones: • Sistema Operativo: Windows XP o Windows 7 • Memoria RAM: 4 GB • Capacidad: 300 MB de disco duro • Tarjeta gráfica: 256 MB		

Tabla 59: RSRec-02. Entorno de producción.

3.1.5 Diagrama de Actividad

Una vez especificados los requisitos del software, junto con la información de los requisitos de usuario y con de los casos de uso, se mostrará un diagrama de actividad del sistema, que

indica el comportamiento de este, desde que se inicializa hasta que se cierra y todos los estados intermedios entre estos.

La actividad empieza con la inicialización de la aplicación, esta lo primero que mostrará en pantalla será el menú principal. En este, el usuario podrá acceder a los records guardados y posteriormente volver al menú principal. También podrá acceder al menú de configuración del sistema donde podrá habilitar y deshabilitar el sonido, elegir el personaje con el que jugará la partida, la dificultad de esta (fácil, normal o difícil) y ver la configuración de teclas con las que controlará al personaje. Una vez finalizada la configuración, volverá de nuevo al menú principal. Para empezar a jugar, desde el menú principal podrá comenzar una nueva partida o acceder al menú de cargar partida, donde aparecerán las partidas guardadas y tendrá que elegir la que desea cargar o volver de nuevo al menú principal. Una vez que está jugando, en cualquier momento el usuario podrá poner en pausa la aplicación apareciendo un menú de pausa, desde el cual podrá reiniciar el nivel, guardar la partida, o salir de esta y volver al menú principal. Hay un tiempo asignado para superar cada nivel, si este se acaba, se le indicará al jugador dándole las opciones de abandonar la partida y volver al menú principal o intentar de nuevo el nivel. Si por el contrario supera el nivel antes de que se acabe el tiempo, en función de si ha conseguido una puntuación de record o no, se mostrará una pantalla con esta información y pasará al siguiente nivel. Así sucesivamente hasta completar el juego. Por último desde el menú principal, saldrá de la aplicación.

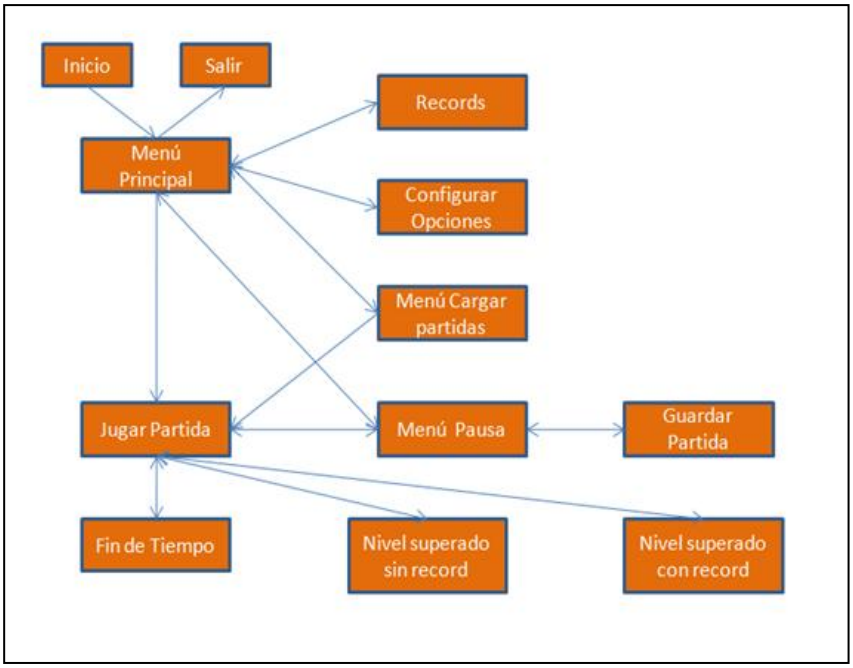

Figura 32: Diagrama de actividad.

3.1.6 Diagramas de Secuencia

Con los diagramas de secuencias se muestra a través del tiempo las relaciones e interacciones entres las diferentes clases u objetos que forman el sistema. A diferencia de los diagramas de caso de uso, se meten en detalles de implementación del nivel, de las entidades y de las clases que compondrán la aplicación.

No se van a representar todos los diagramas de secuencias correspondientes a cada caso de uso, que sería lo normal en la realización de un proyecto, ya que para simplificar se harán tres que servirán de referencia para el resto de casos de uso.

En el diagrama que se observa en la figura 33 se representa la secuencia de opciones del juego. De tal manera que se definen los casos de uso, CU-01 (inicializar aplicación), CU-02 (configurar opciones), CU-03 (música), CU-04 (personaje), CU-05 (dificultad), CU-06 (controles), CU-07 (navegación de menús) y CU-13 (salir aplicación).

Para ello se han definido varios procesos, el primero sería el "*proceso básico*", que es el que se encarga de inicializar la aplicación, arrancar el motor, y llamar a las funciones necesarias para que se ejecute el juego. Posteriormente se ha definido el "*proceso principal*", este hace de núcleo del sistema. Se encarga de crear todas las variables e inicializarlas, llama a toda las funciones necesarias para que pueda aparecer en pantalla la aplicación y el usuario pueda empezar a interactuar con esta, y gestiona todas las interacciones entre los diferentes objetos y componentes del sistema. Finalmente se define el *"proceso o funciones de opciones E/S*", que el que se encarga de gestionar las funciones de entrada y salida de los periféricos ratón y teclado, también genera los menús y controla las opciones que va configurando el usuario.

(1) El proceso básico, inicializa la aplicación, cargando el motor y llamando al proceso principal. (2) Este inicializa todas las variables que tiene la aplicación y llama a las funciones de entrada y salida, para que se habilite el uso del ratón y que se muestre el menú principal. Cuando el usuario pulsa el botón de opciones, el núcleo de la aplicación llama a las funciones de E/S para que se (3) deje de mostrar el menú principal y se (4) muestre el menú de opciones. Aquí el usuario configura (5) la música, la dificultad del juego, elige el personaje y ve los controles para manejar dicho personaje. Una vez que termina de configurar las opciones, sale de este menú (6) y el núcleo del programa vuelve a ejecutar las funciones de E/S para que se vuelva a mostrar el menú principal (7). Finalmente el usuario pulsa el botón salir del menú

principal (8) y el núcleo del programa llama al proceso básico para que se cierre la aplicación (9).

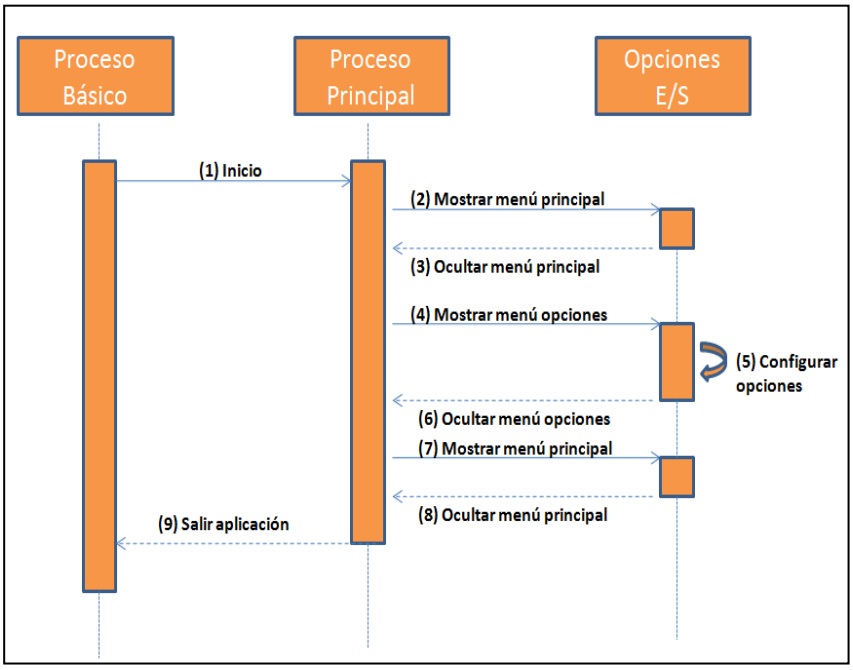

Figura 33: Diagrama de secuencia de opciones.

En el diagrama que se observa en la figura 34 se representa la secuencia de jugar una partida. De tal manera que se definen los casos de uso, CU-01 (inicializar aplicación), CU-07 (navegación de menús), CU-10 (jugar), CU-11 (control personaje) y CU-13 (salir aplicación).

Para ello además de los procesos definidos en el anterior diagrama, se ha definido el proceso "*control entidades*". Este se encarga de gestionar las interacciones del usuario con el sistema a la hora de controlar al personaje, de tal manera que cuando el usuario pulsa la tecla apropiada en el teclado, el personaje se moverá en la dirección indicada. También gestiona, las acciones del personaje con respecto a otros objetos durante el juego, es decir, cuando el personaje entra en contacto con una caja, que este la empuje y la caja se mueva en la dirección correcta.

(1) El proceso básico, inicializa la aplicación, cargando el motor y llamando al proceso principal. (2) Este inicializa todas las variables que tiene la aplicación y llama a las funciones de entrada y salida, para que se habilite el uso del ratón y que se muestre el menú principal. Cuando el usuario pulsa el botón de iniciar una partida nueva, el núcleo de la aplicación llama a las funciones de E/S para que se (3) deje de mostrar el menú principal y que el motor 3D (4) renderice el nivel del juego, coloque al personajes y las cajas en este. A continuación, llama a

las funciones del control de entidades para que el usuario pueda empezar a manejar al personaje y jugar la partida (5). Una vez que termina la partida (6), el proceso principal vuelve a llamar a las funciones de entrada y salida, para que se muestre el menú principal (7). Finalmente el usuario pulsa el botón salir del menú principal (8) y el núcleo del programa llama al proceso básico para que se cierre la aplicación (9).

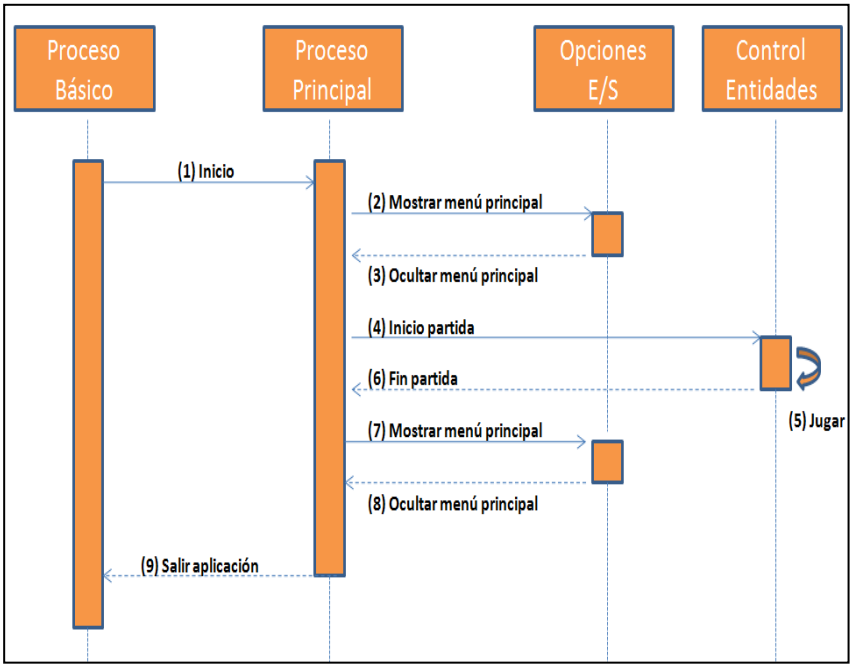

Figura 34: Diagrama de secuencia de jugar partida.

Por último en el diagrama que se observa en la figura 35 se representa la secuencia de establecer records. De tal manera que se definen los casos de uso, CU-01 (inicializar aplicación), CU-07 (navegación de menús), CU-10 (jugar), CU-11 (control personaje), CU-12 (establecer record) y CU-13 (salir aplicación).

Para ello además de los procesos definidos en el anterior diagrama, se ha definido el proceso "*datos*". Este se encarga de gestionar la apertura, lectura, escritura y guardado del fichero en el que se almacenan los datos de las partidas guardadas y los records conseguidos.

(1) El proceso básico, inicializa la aplicación, cargando el motor y llamando al proceso principal. (2) Este inicializa todas las variables que tiene la aplicación y llama a las funciones de entrada y salida, para que se habilite el uso del ratón y que se muestre el menú principal. Cuando el usuario pulsa el botón de iniciar una partida nueva, el núcleo de la aplicación llama a las funciones de E/S para que se (3) deje de mostrar el menú principal y que el motor 3D (4) renderice el nivel del juego, coloque al personajes y las cajas en este. A continuación, llama a

las funciones del control de entidades para que el usuario pueda empezar a manejar al personaje y jugar la partida (5). Una vez que el usuario termina el nivel, se verifica y que la puntuación obtenida es superior a alguno de los records, por tanto se deja la partida (6) y el proceso principal llama a las funciones de entrada y salida para que se muestre por pantalla que ha conseguido un record y que introduzca el nombre para guardarlo (7). El usuario escribe el nombre (8) y da a guardar (9), es entonces cuando las funciones de datos sobrescriben en el fichero donde se guardan las mejores puntuaciones este nuevo record (10). Una vez finalizado se deja de mostrar el menú de record (11) y se continúa con la partida (12). Una vez que termina la partida (14), el proceso principal vuelve a llamar a las funciones de entrada y salida, para que se muestre el menú principal (15). Finalmente el usuario pulsa el botón salir del menú principal (16) y el núcleo del programa llama al proceso básico para que se cierre la aplicación (17).

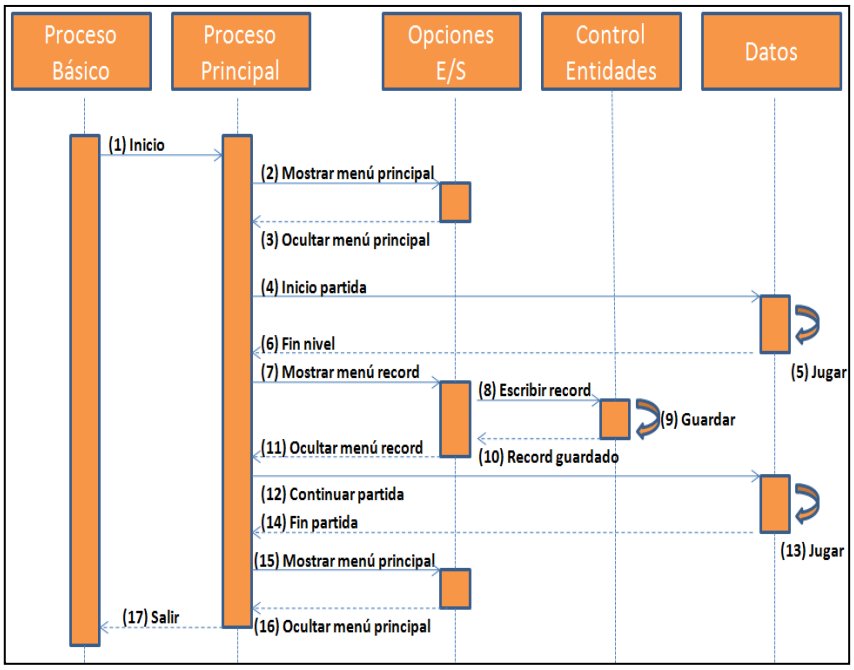

Figura 35: Diagrama de secuencia de establecer record.

3.2 Diseño

A partir de los datos recabados en la toma de requisitos y las conclusiones sacadas de toda la fase previa de análisis, se pasa a la siguiente fase del proyecto, el diseño de la aplicación. En esta se entra en más detalle de las características de la aplicación para que a la hora de la implementación se tenga toda la información necesaria y el resultado final se ajuste lo máximo posible a la aplicación deseada por el cliente.

Primeramente se va a analizar cuál será la arquitectura del software de la aplicación. Posteriormente se mostrarán unos primeros bocetos de como se quiere que sean las pantallas que mostrarán los diferentes menús del juego. También se indicará cómo se quieren que sean los niveles, los personajes y las cajas. A continuación se verá la estructura de carpetas de la aplicación. Y por último, puesto que el lenguaje de programación es lite-c se van a explicar las funciones necesarias que se deben implementar.

3.2.1 Arquitectura

En este punto se presenta como se ha planteado la arquitectura del software de esta aplicación, esta se puede dividir en tres grandes bloques que son:

Datos, gestionan el proceso de carga inicial de la información guardada previamente por el usuario. Es decir, al arrancar la aplicación lo primero que habrá es una carga de los datos almacenados en un fichero que tienen que ver con las partidas guardadas y puntuaciones obtenidas. Durante todo el uso de la aplicación, se podrán modificar estos datos y guardarlos en dicho fichero.

Núcleo de la aplicación, es el que se encarga de inicializar la aplicación, y gestionar tanto la carga de niveles como la del usuario, el renderizado de toda el mundo 3D, y por supuesto todo el control del juego.

Finalmente el último bloque es el de Entrada/Salida, este gestiona o hace de interfaz de usuario entre la aplicación y el usuario, mediante las entradas de teclado y ratón, y salidas tales como visualización de menús.

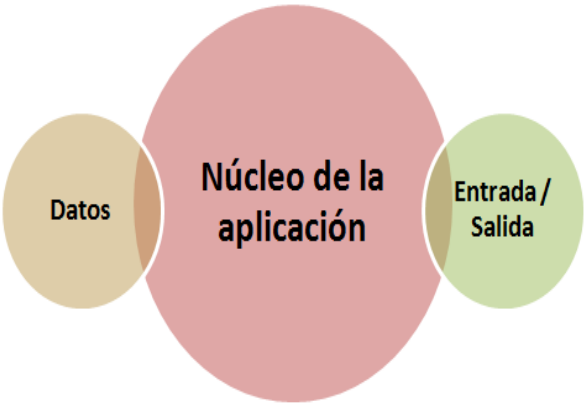

Figura 36: Arquitectura del software

3.2.2 Pantallas

A continuación se presentan unos bocetos de como se plantean que serán las pantallas, pero más a nivel funcional que a nivel de diseño.

En la figura 37 se muestra una idea de cómo debe ser el menú principal. Esta será la primera pantalla que mostrará la aplicación. En esta tiene que aparecer el título del juego, "*Sokoban*", y debajo cuatro botones. Un botón que al pulsarlo inicie automáticamente una nueva partida con las opciones que estén configuradas por defecto o que haya modificado previamente el usuario. Otro botón, records, debe llevar al usuario a otra pantalla en el que aparecerán los records guardados. Un tercer botón, que lleve al usuario a una nueva pantalla para cargar una partida guardada previamente. Y por último un cuarto botón, en el que se acceda al menú de configuración de opciones.

Figura 37: Pantalla menú principal.

En la figura 38 se muestra una idea de cómo debe ser el menú records. Esta pantalla la encabezará un título que ponga "*records*". A continuación se mostrarán las cinco mejores puntuaciones, ya que la aplicación no va a guardas más puntuaciones que las cinco primeras. En cada línea se mostrará la posición, a continuación el nombre con el que se guardó el record, seguido de una línea de puntos hasta el final de la línea, donde aparecerá la puntuación. Debajo de las puntuaciones aparecerá un botón, con la opción de volver. Al pulsarlo se volverá al menú principal. Esta pantalla no ofrecerá más funcionalidades que ver los records y volver al menú principal.

Figura 38: Pantalla de records.

En la figura 39 se muestra una idea de cómo debe ser el menú de cargar partida. Esta pantalla la encabezará un título que ponga "*cargar partida*". Como la aplicación no va a guardar un número ilimitado de partidas, sino que sólo va a permitir almacenar tres, el menú de cargar partida mostrará un botón con cada partida guardada y el nombre con la que se guardó. El usuario a pulsar uno de estos tres botones, hará que se cargue automáticamente la partida en el nivel y con las opciones configuradas con las que se estaba jugando esa partida en el momento que se guardó. Por último se mostrará un botón con la opción de volver por si el usuario finalmente no quiere cargar una partida y desea volver al menú principal.

Figura 39: Pantalla de cargar partida.

En la figura 40 se muestra una idea de cómo debe ser el menú de opciones. Esta pantalla la encabezará un título que ponga "*opciones*". Seguido de este, aparecerán tres líneas con dos botones en cada una. El botón de la izquierda mostrará el nombre de la opción a configurar y el botón de la derecha mostrará la configuración actual de esa opción. De tal manera que si el

usuario pulsa una vez en el botón de la música, cambiará en el botón de la derecha de On a Off. En la segunda línea se elegirá el personaje entre los que habrá disponibles, como mínimo habrá dos diferentes. En la tercera línea se elige la dificultad, entre fácil, normal y difícil. En principio, esto lo que variará es el tiempo que tiene el jugador para superar cada nivel. En la cuarta línea, al pulsar el botón de controles aparecerá una pantalla en la que se mostrará la configuración del teclado para manejar al personaje. Por último se mostrará un botón con la opción de volver para cuando se termine de configurar las opciones y se desee volver al menú principal.

Figura 40: Pantalla de opciones.

En la figura 41 se muestra una idea de cómo debe ser el menú de pausa. Mientras se está jugando una partida, el jugador pulsara una tecla, configurada a tal efecto, para que se pare el juego y aparezca esta pantalla del menú de pausa. Esta pantalla la encabezará un título que ponga "*pausa*". Seguido de este, aparecerán cuatro botones. El primero dará la opción de continuar la partida, de tal manera que se reanudará el juego tal y como se había dejado antes de entrar en pausa. El siguiente botón da la opción de reiniciar el nivel, ya que por la dinámica del juego puede que el jugador llegue a colocar una caja en una esquina con lo que será imposible que la saque de ahí y por tanto no podrá superar el nivel. De esta manera volverá a comenzar la partida en ese nivel pero con todo el tiempo disponible de nuevo, y con el personaje y las cajas colocadas en sus posiciones iniciales. El tercer botón le dará la opción de navegar al menú de guardar la partida. Y por último, el cuarto botón, salir partida, hará que el jugador abandone la partida y vuelva al menú principal. Si no ha guardado la partida previamente perderá todo lo conseguido.

Figura 41: Pantalla de menú de pausa.

En la figura 42 se muestra una idea de cómo debe ser el menú de guardar partida. Como se vio anteriormente, a este se accede a través del menú de pausa. Esta pantalla la encabezará un título que ponga "*guardar partida*". Seguido de este, aparecerán tres líneas con dos botones en cada una. El botón de la izquierda mostrará las tres ranuras disponibles en las que se podrá guardar una partida, cuando el usuario pinche sobre una de estas automáticamente se habilitará el teclado para que escriba el nombre con el que desea guardar la partida. Después para confirmar que quiere salvar la partida, tendrá que pulsar el botón de la derecha. A continuación se mostrara un mensaje por pantalla que indicará que la partida se ha guardado correctamente. . Por último se mostrará un botón con la opción de volver para cuando se termine de guardar la partida y se desee volver al menú de pausa.

Figura 42: Pantalla de guardar partida.

En la figura 43 se muestra una idea de cómo debe ser la pantalla de fin de tiempo. Cuando se está jugando un nivel y se acaba el tiempo asignado para superar dicho nivel, el juego se

termina y aparece una pantalla indicando al jugador que se le acabado el tiempo. Esta pantalla la encabezará un título que ponga "*fin del tiempo*". Seguido de este, se le indicará que se le ha acabado el tiempo para superar esta fase. A continuación le aparecerán dos botones con las dos opciones por las que puede optar el usuario. La primera es reintentar el nivel con lo que volverá a comenzar la partida en ese nivel pero con todo el tiempo disponible de nuevo, y con el personaje y las cajas colocadas en sus posiciones iniciales. La segunda opción será salir de la partida y volver al menú principal.

Figura 43: Pantalla de fin del tiempo.

En la figura 44 se muestra una idea de cómo debe ser la pantalla de nuevo record. Al contrario que en el caso anterior, el jugador ha logrado superar un nivel y no sólo eso sino que lo ha hecho consiguiendo un nuevo record. En este punto la aplicación le mostrará esta pantalla, a la que la encabezará un título que ponga "*nuevo record*". Seguido de este, aparecerá un mensaje dándole la enhorabuena e indicándole los puntos conseguidos y en qué posición se coloca. A continuación, el usuario escribirá su nombre con el que quedará registrado el record. Finalmente se muestra un botón para continuar la partida.

Figura 44: Pantalla de nuevo record.

Mientras se está jugando la partida en la parte superior de la pantalla aparecerá un panel en la que el jugador podrá ver en todo momento el tiempo que le queda para superar el nivel, el número de cajas que ha colocado correctamente y los puntos que lleva conseguidos. La figura 45 muestra una idea de cómo debería ser panel. Sería deseable, que en el campo del tiempo se viera el dibujo de un reloj y en el de las cajas el dibujo de una caja.

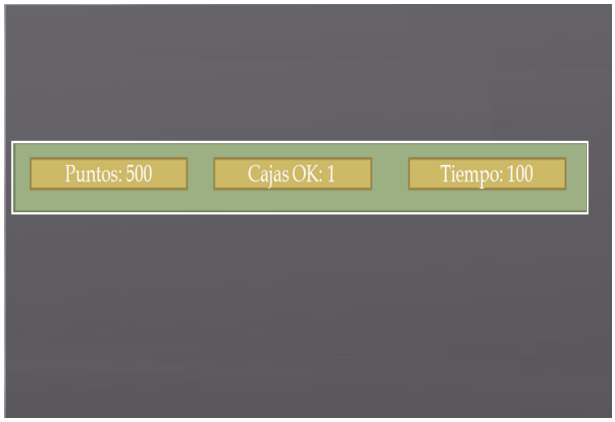

Figura 45: Panel de juego.

También sería deseable que a nivel de diseño gráfico, todas las pantallas que se han comentado anteriormente tuvieran motivos relacionados con el juego. Como por ejemplo, que las letras de los títulos se hagan con cajas pequeñas.

En cuanto a los personajes, se quiere que las cajas sean grandes en proporción al tamaño de estos. En la figura 46 se ve una imagen obtenida de una página de internet {87}, que muestra esta proporción que se quiere dar al tamaño de estas entidades.

Figura 46: Proporción tamaño personaje respecto de las cajas.

Finalmente en la figura 47 se muestran dos capturas de dos juegos del tipo de Sokoban disponibles en la red {88}, {89}, estas sirven de referencia de cómo sean los niveles del juego.

Figura 47: Referencia de diseño de niveles de Sokoban.

3.2.3 Carpetas del proyecto

Este proyecto constará de una carpeta sokoban, que contendrá toda la aplicación. Esta a su vez contendrá otra carpeta con todas las imágenes.

3.2.4 Librerías de 3D Game Studio

En el código del proyecto sólo se utilizarán dos de las librerías de lite-c, estas añaden un archivo de código al programa, y son:

- Acnex.h, es un código estándar que se necesita para poder programar en lite-c y que incluye todas las variables predefinidas y funciones del motor 3D GameSutdio A7.

- Default.c, es un código que contiene varias funciones de uso habitual tales como mostrar la consola de comandos, poder finalizar la aplicación pulsando la tecla escape (Esc).

3.2.5 Archivos del proyecto

Puesto que Lite-c es una versión muy simple y reducida de C++, no se tienen clases definidas como tal, sino que más bien se utilizarán una serie de funciones que implementarán los tres bloques que se comentaron en el punto de arquitectura del software.

Tendremos dos ficheros con extensión .c que contendrán toda la programación de la aplicación.

- Datos.c, en este se implementará todas las funciones para la lectura y modificación de los datos guardados.

• Sokoban.c, en este se implementarán todas las funciones del núcleo de la aplicación y las interfaces de Entrada/Salida con el usuario final. Contendrá la función main o principal que es la que arranca la aplicación.

Se pasa a analizar cada bloque con las funciones necesarias para su implementación:

Datos

Componente:	**Datos**
Variables:	- STRING* nombrePart; - STRING* personajePart; - STRING* puntosPart; - STRING* nivelPart; - STRING* nivelDfPart; - STRING* Puntos; - STRING* Nombre; - var ini_nLinesC; - STRING* partida; - STRING* puntos; - STRING* nivel; - STRING* jugador; - STRING* difi; - STRING* nomRecord; - STRING* puntRecord;
Funciones:	- function Archivo_Leer (STRING* _sSection, STRING* _sKey, STRING* _sValue); - function Archivo_Modificar (STRING* _sSection, STRING* _sKey, STRING* _sValue)); - function Archivo_Cargar (STRING* _sFile); - function Archivo_Guardar(STRING* _sFile); - function CargarDatos(); - function guardarPartida(STRING* nombrePart,STRING* personajePart,STRING* puntosPart,STRING* nivelPart, STRING* nivelDfPart, STRING* ranura); - function cargarPartida(); - function guardarPartidaRanura(); - function guardarRecord(STRING* nombreRecord,STRING* puntosRecord, STRING* posicion);

Tabla 60: Componente: Datos.

Los datos de las partidas y records guardados estarán recogidos en un fichero que se llamará "Data.CFG". Como no se había planteado en los requisitos, en principio estos datos no llevarán una seguridad añadida, aunque si sería recomendable que la tuviera ya que tienen información sensible para la aplicación y que podría hacer que esta no funcionara correctamente.

Cuando se inicialice la aplicación, esta llamará a la función de "*CargarDatos*", que se encargara de llamar a las funciones necesarias para leer este archivo y asignar los valores leídos a una unas variables globales a las que se podrá acceder, leer y modificar en cualquier momento. Cuando un jugador quiera cargar una partida no será necesario volver a leer el archivo Data, ya que están todos los datos necesarios en estas variables.

También cuando el usuario quiera guardar una partida, una vez elija la ranura deseada la aplicación ejecutara la función "*guardarPartidaRanura*", que se encargará de pasar la información de las variables globales necesarias a las funciones que se encargan de modificar el archivo de datos.

Entrada/Salida

Componente:	**Entrada/Salida**
Variables:	- SOUND* SND_MENU; - SOUND* SND_JUEGO; - SOUND* SND_GAME_OVER; - SOUND* SND_SCORE; - SOUND* SND_CAJA; - SOUND* SND_BONUS; - var punteroMusica; - var camara; - BMAP* back_ground; - BMAP* bk_Pausa; - BMAP* bk_Guardado; - BMAP* bk_Records; - BMAP* bk_nuevoRecord; - BMAP* bk_Controles; - BMAP* bk_Fin"; - BMAP* bk_completado; - PANEL* panelInicio; - PANEL* panelPausa;

	- PANEL* panelFin; - PANEL* panelNivelCompletado; - PANEL* panelJuegoSuperado; - PANEL* panelGuardar; - PANEL* panelCargar; - PANEL* panelRecords; - PANEL* panelEscribirRecords; - PANEL* panelPartidaGuardada; - PANEL* panelOpciones; - PANEL* panelControles; - PANEL* panelJuego; - TEXT* opciones_txt; - TEXT* guardarRecord_txt; - TEXT* records_txt; - TEXT* posiciones_txt; - TEXT* puntuaciones_txt;
Funciones:	- function cambiarDificultad(); - function cambiarJugador(); - function cambiarMusica(); - function controlSonido(var sonido); - function cargarPartida(); - function continuarPartida(); - function escribirRanura(); - function escribirRecord(STRING* posicion); - function mostrarCargar(); - function mostrarControles(); - function mostrarGuardar(); - function mostrarOpciones(); - function mostrarPausa(); - function mostrarRecords(); - function ocultarMenus(); - function posCamara(); - function posicion_camara(ENTITY* ent); - function volverCargar(); - function volverGuardar(); - function volverOpciones(); - function volverRecords();

Tabla 61: Componente: Entrada/Salida.

Entrando un poco más en detalle, se explican cómo se va a controlar la cámara, el sonido, la navegación de menús y la entrada de datos.

Empezando por la cámara, se creara un proceso paralelo a la aplicación que controla en todo momento la posición de la cámara. Este proceso arranca en el momento en que es creada la entidad del personaje, que cómo se verá en la implementación, tiene una acción asignada que llamará a la función "*pos_camara*". Esta función, según del valor de la variable cámara, ajustar los valores de esta para que sea una vista en tercera persona, una vista desde arriba de todo el nivel, o una vista desde arriba pero más cercana que la anterior que se mueve con el movimiento del personaje.

Para el control del sonido, se tendrá otro proceso paralelo a la aplicación que se iniciará en la inicialización de la aplicación, para el control de este se hará a través de las variables globales "*punteroMusica*" y "música" a las que se podrá acceder, leer y modificar en cualquier momento.

Para la navegación de menús en lite-c, se trabajará con variables del tipo PANEL, que tienen un una imagen de fondo y que se superponen a la imagen del juego en función de sus valores de layer o capa, de tal manera que variando los valores de su capa o con funciones que varían un flag que tienen como atributo hacen que sean visibles o no. De ahí que se tenga un panel para cada menú que vaya a aparecer en pantalla.

Y finalmente para la entrada de datos desde el teclado, se tendrán variables del tipo TEXT que muestran un texto en pantalla y que se pueden modificar y por tanto se modifica el mensaje que aparece en pantalla.

Núcleo

Componente:	**Núcleo**
Variables:	- ENTITY* jugador; - ENTITY* caja1; - ENTITY* caja2; - ENTITY* caja3; - ENTITY* caja4; - ENTITY* caja5; - ENTITY* caja6; - ENTITY* reloj; - var cajas_ok; - var score;
Funciones:	- function borrarEntidades(); - function caja_colocada(var prux, var pruy); - function calcularDistancia(var x, var y); - function calcularPuntuacion(); - function esRecord(); - function fin_juego(); - function inicializar_nivel(); - function iniciarPartidaCargada(); - function mover_caja(); - action caja_quieta(); - action contador(); - action jugador_andando();

Tabla 62: Componente: Núcleo

Finalmente se analiza el diseño de este componente principal de la aplicación, como es el núcleo, que va a gestionar el grueso del juego. Este se va desarrollar con entidades, cada entidad es un hilo o un proceso que corre en paralelo con la aplicación. A cada entidad además se le puede asignar una acción para que ejecute el código que se crea conveniente. De las entidades que se han considerado necesarias se pueden separar en tres grupos, que serían el jugador, por otro lado las cajas y finalmente el reloj. A continuación se analizarán un poco más en detalle.

La entidad jugador, será el personaje que manejará el usuario, esta se creara en cada inicialización de nivel y se eliminará cuando se finalice dicho nivel. Esta recogerá las pulsaciones del teclado para mover al personaje por la pantalla.

En cuanto a las cajas, se ha decidido crear una entidad por cada caja y que por el motor de colisiones del 3D Game Studio, detecten cuando las empuja el personaje. En el momento que detecten dicha colisión, se moverán con el movimiento del jugador hasta que deje de empujarla. Una vez deje de moverse la caja verificará su posición y si está dentro de la zona de descarga, de ser así comprobara si ya están todas las cajas colocadas para ver si se ha superado el nivel. Estas entidades, al igual que la del personaje, se crean y se destruyen con la inicialización y finalización de un nivel.

Finalmente se creara la entidad reloj, que será la encargada de gestionar el tiempo de juego de la partida. También se crea y destruye con el nivel.

3.3 Implementación

Finalmente se llega a la fase de implementación, en la que se lleva a la realidad el trabajo realizado en las fases anteriores de análisis y diseño. A continuación no se va a entrar en detalle de todo lo que se ha implementado sino más bien en los detalles más relevantes.

Previa a la realización del proyecto se han hecho una serie de tutoriales, disponibles en la página web de 3D Game Studio {8},**¡Error! No se encuentra el origen de la referencia.** para aprender el manejo de la aplicación y que son de muy recomendable utilización antes de enfrentarse por primera vez al desarrollo de una aplicación con este motor gráfico.

De las varias aplicaciones que incorpora la versión gratuita del 3D Game Studio, para la implementación del videojuego se han utilizado las siguientes:

- El motor gráfico A7.
- Lite-C, lenguaje de programación que soporta.
- MED, editor de modelos y terrenos. A partir de este se han creado los modelos de los personajes y de las cajas.
- SED, editor de programación, depurador y compilador. Con esta aplicación se ha programado todo el juego

- WED, entorno de desarrollo de niveles. A partir de este se han creado los tres niveles que va a tener el juego.

A continuación se analizará cómo se han usado estas herramientas a la hora de realizar la implementación de la aplicación. Aunque se van a ver las tres últimas mencionadas, la memoria del proyecto se centrará principalmente en las aplicaciones SED y WED, que han sido las herramientas principales de esta fase.

3.3.1 MED (Model Editor)

Desde el punto de vista del mundo 3D, se entiende por un modelo a una representación visible de un conjunto de objetos, que una vez procesados o renderizados {90}, se convertirán en una imagen 3D. Estos modelos pueden ser estáticos, como una pared o pueden ser dinámicos como el personaje de un juego.

Figura 48: Ejemplo de modelos estáticos y dinámicos de 3D.

La técnica más habitual de modelado consiste en hacer una malla poligonal, de tal manera que a base de unir pequeños polígonos de diferentes tipos se da forma al objeto. En la figura 49 se puede ver un ejemplo del modelado de una persona. Como se puede observar hay tres modelos diferentes de una misma mujer, la diferencia fundamental está en el número de polígonos para darla forma. Un mayor número de polígonos permite un modelo con mucha mayor calidad gráfica y que se asemeje más a la realidad. Pero tiene como consecuencia un mayor coste computacional, es decir, para poder representar este modelo un ordenador necesita hacer muchas más operaciones y cálculos matemáticos, lo que exige que sea más potente y que tenga una tarjeta gráfica de alta gama. De ahí, que el modelado juegue un papel muy importante en el diseño de un videojuego, ya que además del aspecto gráfico de este, influirá directamente en los requisitos del hardware que debe cumplir la plataforma en la

que se desee usar dicha aplicación. Así un juego de última generación no podrá ser soportado nunca por un ordenador de bajas prestaciones.

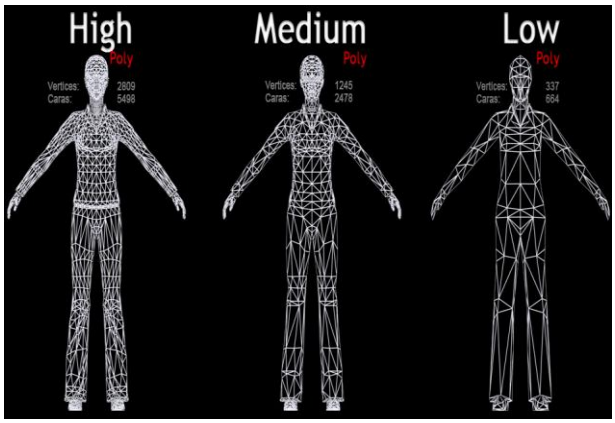

Figura 49: Ejemplo de modelado poligonal.

En el mercado actual se puede encontrar un gran número de aplicaciones de modelado 3D, en la tabla 63, se muestran unos cuantos ejemplos.

Nombre	Compañía	Enlace
Maya	Autodesk (antes alias|wavefront)	http://www.autodesk.com/maya
SOFTIMAGE|XSI	Autodesk (antes propiedad de AVID y antes de Microsoft)	http://www.softimage.com
3DStudio MAX	Autodesk	http://www.autodesk.com/3dsmax
LightWave	Newtek	http://www.newtek.com/
Blender	Blender (OpenSource)	http://www.blender.org/
Cinema 4D	Maxon	http://www.maxon.net
Houdini	Side Effects	http://www.sidefx.com/
Rhinoceros	Rhino	http://www.rhino3d.com/
Pov-ray	Povray	http://www.povray.org/
Cheetah 3D	Cheetah 3D	http://www.cheetah3d.com/
ZBrush	Pixologic	http://www.pixologic.com/home.php

Tabla 63: Ejemplos de aplicaciones de modelado 3D.

En cuanto a MED, se puede decir es una herramienta gratuita para crear, editar y convertir modelos y terrenos 3D. Aunque no se puede comparar con el resultado final que pueden proporcionar otras aplicaciones como las ya mencionadas 3ds Max {91} y Maya, es una herramienta potente que incluye animación de "huesos" y soporta el manejo de sombras. También permite importar modelos de otras aplicaciones de modelado 3D como 3ds Max.

No es objetivo de este proyecto dar una disertación de modelado 3D ya que sólo esta parte podría ser otro proyecto en sí, de hecho para la implementación del juego no se ha llegado a modelar ningún personaje sino que se ha trabajado con unos modelos gratuitos de la página de Acnex {92}. Primeramente se verá el interfaz que proporciona esta aplicación, y posteriormente se verá más detalladamente dos herramientas fundamentales, que proporciona MED, que son la animación de huesos "*bones animation*" y su utilidad práctica. Y como aplicar una textura a un modelo, mediante el skin editor.

En la figura 50 se puede ver una captura de MED. Tiene un menú superior con todas las herramientas que proporciona esta aplicación, también tiene 4 ventanas que se corresponden con las vistas de alzado, perfil y planta del nivel y una cuarta que es el modelo terminado en tres dimensiones.

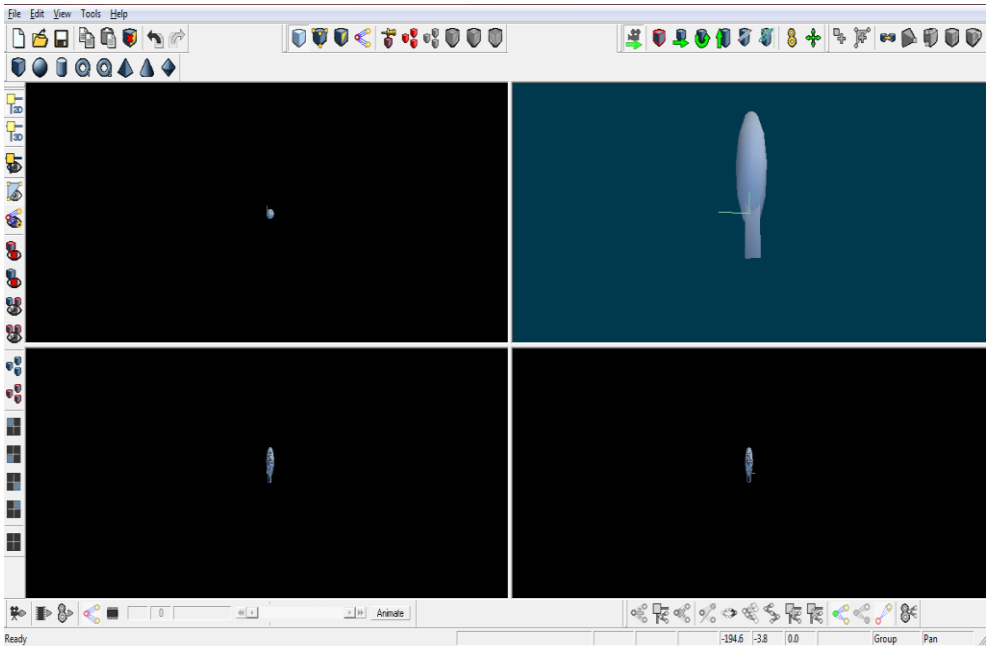

Figura 50: Captura de MED.

En la tabla 64 se explican brevemente las funciones de los botones que aparecen en la barra de herramientas de MED.

	Crear nuevo proyecto med.
	Abrir un proyecto med existente.
	Guardar proyecto med.

	Mover un modelo.
	Rotar un modelo sobre el eje seleccionado.
	Modificar el tamaño de un modelo manteniendo sus proporciones.
	Permite al usuario moverse con el ratón por las diferentes vistas,
	Crear un cubo
	Crea una esfera.
	Crea un cilindro.
	Crea un toroide.
	Crea una pirámide.
	Crea un cono.
	Crea un cono doble.
	Eliminar el objeto seleccionado.
	Muestra los vértices de los polígonos que forman el modelo.
	Muestra las caras de los polígonos que forman el modelo.
	Activa el "*bone mode*".
	Crea un nuevo hueso.
	Muestra si un vértice de un polígono está asociado a un hueso o no.
	Asocia los vértices que se seleccionen a un hueso.

Tabla 64: Barra de herramientas de MED.

3.3.1.1 Bones animation

En cualquier videojuego se quiere que los modelos hagan movimientos que se asemejen a la realidad. Por ejemplo, si se tiene un modelo de una persona, se quiere que al desplazarse por el nivel, esta ande moviendo las piernas y que el cuerpo siga el movimiento con la mayor naturalidad posible y no que simplemente se cambie de posición sin haber movido ninguna parte de su cuerpo. En este punto es donde entra bones animation (animación de huesos), esta técnica consiste en dotar de esqueleto a un modelo ya creado previamente.

Un hueso se puede considerar como una figura geométrica, a la que se le asocian un determinado número de vértices, correspondientes a los polígonos del modelo. En el momento que se produce dicha asociación, el hueso tiene una serie de atributos tales como su posición dentro del modelo, su longitud, sus ángulos de orientación. De esta manera, si se rota o se mueve este hueso, se mueven y se rotan los vértices asociados a este. Para ilustrarlo, se puede comparar con el esqueleto humano, a un hueso están ligados una serie de músculos y de piel, cuando se mueve este hueso, se mueven todos sus músculos y piel.

En el caso de modelos de personas, el esqueleto se asemeja bastante al del cuerpo humano real, aunque no tiene por qué tener todos los huesos, ya que no son necesarios para que un modelo pueda hacer cualquier tipo de movimiento. Pero el uso de esta técnica no es sólo para modelos de personas, es válido para cualquier modelo al que se le quiera dar movimiento.

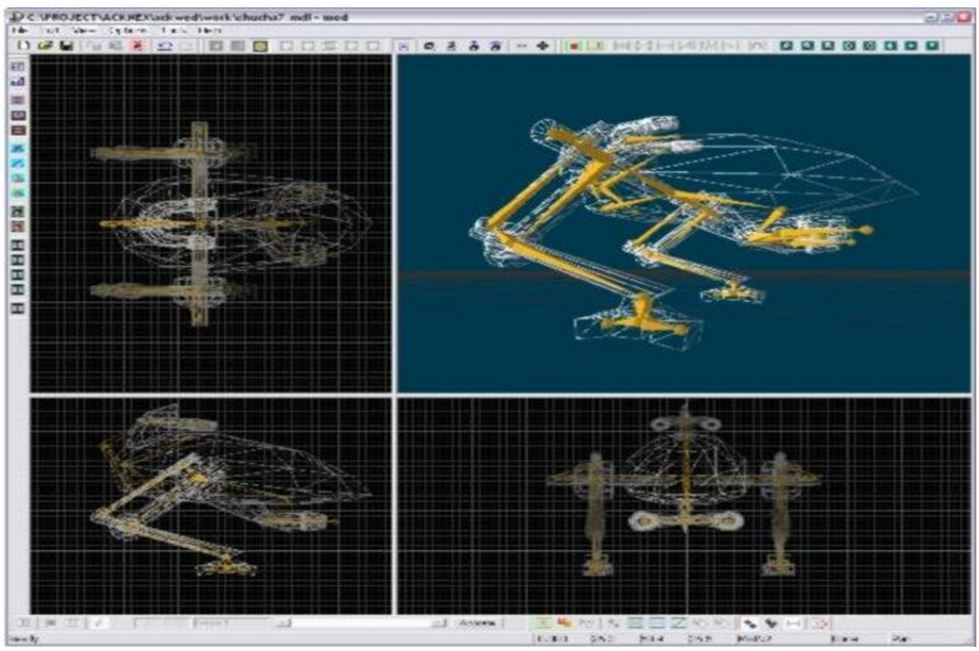

Figura 51: Robot animado con MED.

Un ejemplo se muestra en la anterior figura 51, en esta se puede observar un robot al que se le ha dotado de un esqueleto. Los huesos son esos objetos o figuras amarillas, que se encuentran en las patas y parte de la cabeza. Analizando la imagen, ya se puede saber que este robot podrá mover las patas, los pies y el cañón que se encuentra en la parte inferior de la cabeza.

Con MED es bastante sencillo crear el esqueleto de un modelo, brevemente se enumeran los pasos a seguir (todos los botones que se usan a continuación están indicados en la tabla 64):

1. Lo primero y fundamental es tener un modelo creado.
2. Se activa el "*bone mode*" pulsando el botón de la barra de herramientas que está destinado a tal efecto. Esto hace que se habiliten una serie de botones situado en la parte inferior de la pantalla,
3. Se pulsa el botón crear nuevo hueso.
4. En cualquiera de las vistas se pulsa en la posición en la que se quiera colocar.
5. Se le da el tamaño y se ajusta la posición deseada.
6. Se siguen creando todos los huesos necesarios hasta formar el esqueleto. A cada hueso se le debe asignar un nombre.
7. Una vez creado el esqueleto, se muestran todos los vértices del modelo pulsando el botón de la barra de herramientas destinado a tal efecto.
8. Se seleccionan los vértices que se quieren asociar a cada hueso.
9. Se pulsa el botón de asociar vértices. De esta manera ya se tienen unidos el esqueleto y la malla que forma el modelo.

Llegados a este punto ya se tiene un modelo con su esqueleto, ahora toca hacer que se mueva. Para conseguir esto, se puede hacer desde la propia programación del código del juego o utilizando MED.

En el primer caso se usarían una serie de funciones predefinidas que permiten actuar sobre los huesos de un modelo. Por ejemplo, la función " *ent_bonerotate*", rota el ángulo del hueso

del modelo que se le pase por parámetro. En el caso del robot de la figura 51 se podría programar un código que llamará a esta función e hiciera que rotará el ángulo del hueso que se corresponde con el cañón, de esta manera podría hacer que apuntara en otra dirección. El gran inconveniente que tiene este primer método, es que sólo es práctico para movimientos muy sencillos, sería muy costoso mover todos los huesos del modelo de una persona para hacer que ande. De ahí que en la implementación de este proyecto se ha optado por usar MED para mover los modelos.

Esta segunda manera de hacer que un modelo se mueva es mucho más intuitiva que la primera, ya que se basa en un concepto ampliamente conocido y es el del video en movimiento. Para que el ser humano llegue a ver un objeto en movimiento, las imágenes que aparezcan en la pantalla deben mostrarse a una determinada velocidad de frames por segundo. A 12 frames por segundo, se pueden detectar ligeros parpadeos y a partir de 70 frames por segundo no mejora la sensación de movimiento, con lo que la velocidad debe estar entre 12 y 70 frames por segundo. Las películas que se pueden ver en los cines se muestran a 24 frames por segundo. Esto lo que quiere decir, es que si se quiere mostrar un movimiento de un brazo moviéndose desde la cintura hasta la cabeza que duré un segundo, en pantalla deben aparecer seguida una tras otra, 24 fotos del brazo según se va moviendo.

Luego si se quiere hacer que un modelo se mueva, se tendrá que hacer una foto de este en una posición, mover ligeramente el modelo manipulando los huesos de su esqueleto y hacer otra foto, y repetir este proceso hasta que se termine el movimiento.

En la figura 52, se muestra uno de los dos modelos usados en la aplicación, el modelo del vikingo. En la imagen de la izquierda se observa al modelo finalizado y en la imagen de la derecha se puede ver el esqueleto con el que se le ha dotado.

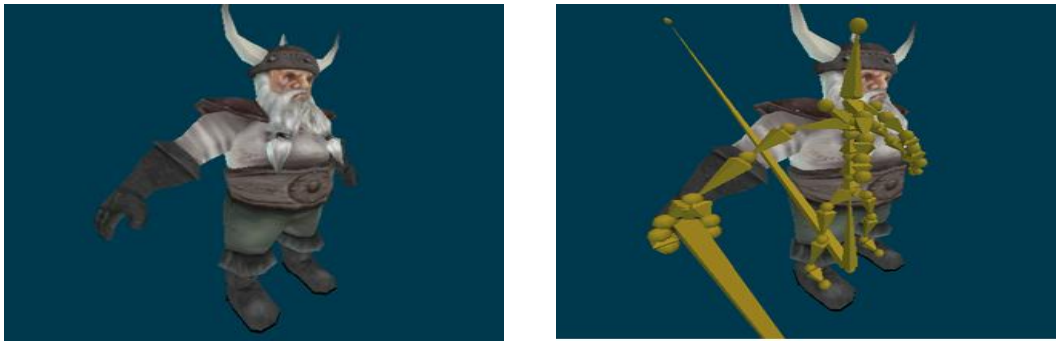

Figura 52: Modelo del vikingo.

Para realizar todo este proceso de dar movimiento a este modelo, MED tiene una herramienta llamada "*Manager Frame*", que es una función que se encuentra en el menú de edición. El proceso es muy sencillo, se parte de una posición inicial, se mueven los huesos del vikingo hasta que inicien parte del movimiento, se da al botón de "*add frames*" para capturar esa imagen y se le da un nombre, y se repite esto hasta que se termine todo el movimiento. Es muy importante que cada captura tenga el mismo nombre seguido del número de frame que es. De tal manera que para cada tipo de movimiento se tenga una secuencia de frames específica. En la figura 53 se puede ver que el vikingo tiene dos tipos de movimientos que son andar y correr.

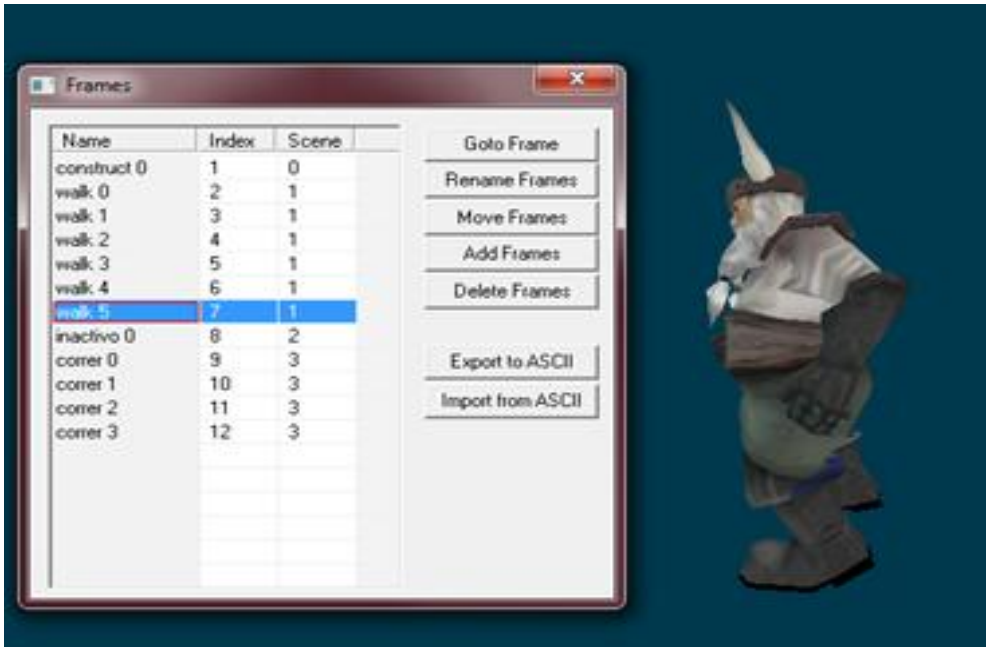

Figura 53: Modelo vikingo, manage frames.

Para el desarrollo del juego basta con que el personaje ande por el nivel, con lo que no se van a implementar más movimientos. Aunque como se ha visto no sería complicado hacer que el modelo saltara, se agachara o se tumbara. En la figura 54 se muestra el ciclo completo de capturas para que el modelo ande por el nivel. Posteriormente se verá cómo se programa en el código de la aplicación para que el personaje se mueva de esta manera, cuando el usuario esta pulsando las teclas que controlan al vikingo.

Figura 54: Modelo vikingo, ciclo completo de movimiento de andar.

3.3.1.2 Skin editor

Esta es la segunda función que se quería destacar de esta aplicación. Como su nombre bien indica es un editor de la "*piel*" del modelo. Como ya se había visto en los anteriores puntos, cuando se crea un modelo lo que se tiene en primera instancia es una malla poligonal pero no tiene "piel", no tiene color, ni rasgos que hagan que se asemeje a la realidad. En la figura 55 se muestra como es el modelo antes de darle una piel:

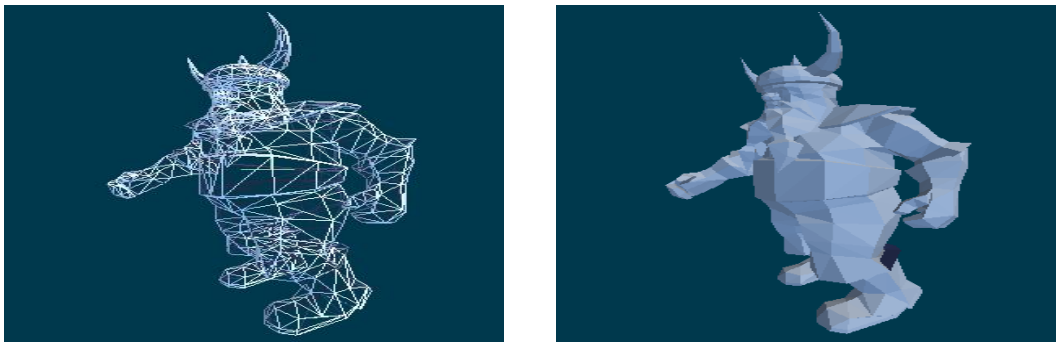

Figura 55: Modelo poligonal del vikingo.

Lo que hace el skin editor es asignar una "piel" o capa a los diferentes vértices del modelo. Esta función se ejecuta desde el botón file. En la figura 55 se muestra el interfaz que ofrece la aplicación al respecto. En la izquierda se muestra todos los vértices del modelo y la piel que se le ha asignado a cada vértice. En la pantalla de la derecha se puede ver el resultado final.

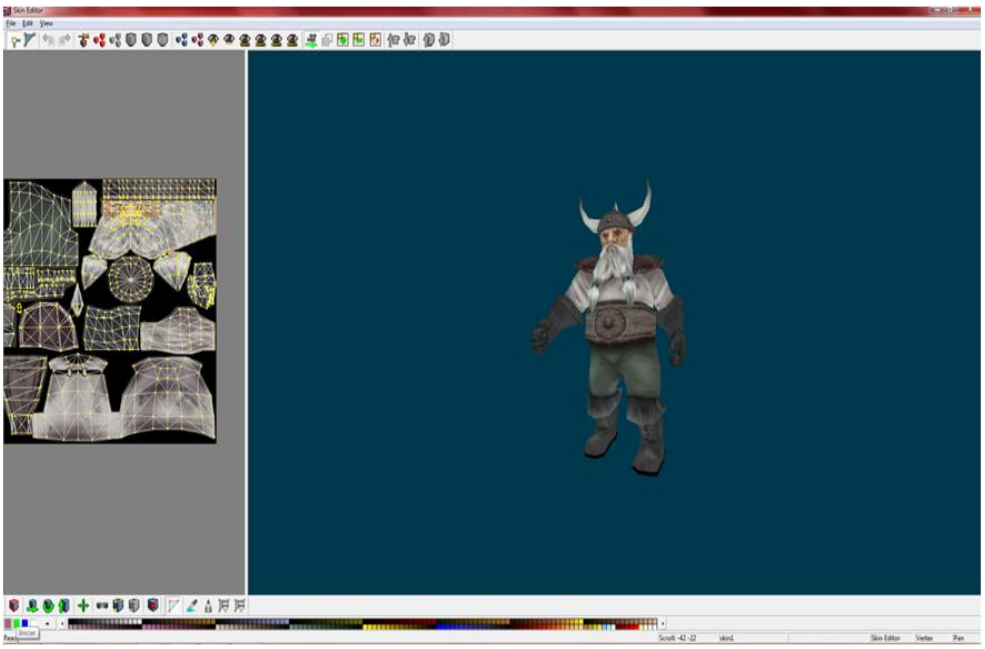

Figura 56: Skin editor, modelo vikingo.

Puesto que no se han llegado a diseñar modelos propios de los personajes para el juego, esta herramienta no se ha llegado a usar en profundidad por lo que no se va a entrar en más detalle de cómo funciona. Simplemente se va a mencionar como se han creados las cajas, para ello se descargó de la página de Acnex {92}, el modelo de un cubo de un juego espacial. Como la piel que tenía era muy futurista se cambió la imagen de esta por otra descargada de internet. Para cambiar la imagen se uso Skin editor, esta herramienta permite importar imágenes desde File-> Import->Skin Image.

3.3.2 WED (Level Editor)

Es el editor para crear mundos virtuales 3D o niveles. También sirve de centro de control ya que se pueden asociar a los niveles creados scripts y modelos. Es una herramienta intuitiva, que viene con una serie de texturas y permite importar y añadir otras. Para la creación de los niveles sólo se han utilizado texturas que vienen con la aplicación o que se puedan descarga de la página de Acknex {92}.

Como ya se ha indicado anteriormente, se han implementado 3 niveles, a continuación se va a analizar cómo se ha creado el primero y se mostrará el resultado final de los otros dos.

Pero antes de ver los pasos seguidos para la creación del primer nivel, se explicará brevemente el manejo de esta herramienta para que sea más comprensible la implementación seguida en el proyecto.

En la figura 57 se puede ver una captura de Wed. Tiene un menú superior con todas las herramientas que proporciona esta aplicación, también tiene 4 ventanas que se corresponden con las vistas de alzado, perfil y planta del nivel y una cuarta que es el nivel terminado en tres dimensiones.

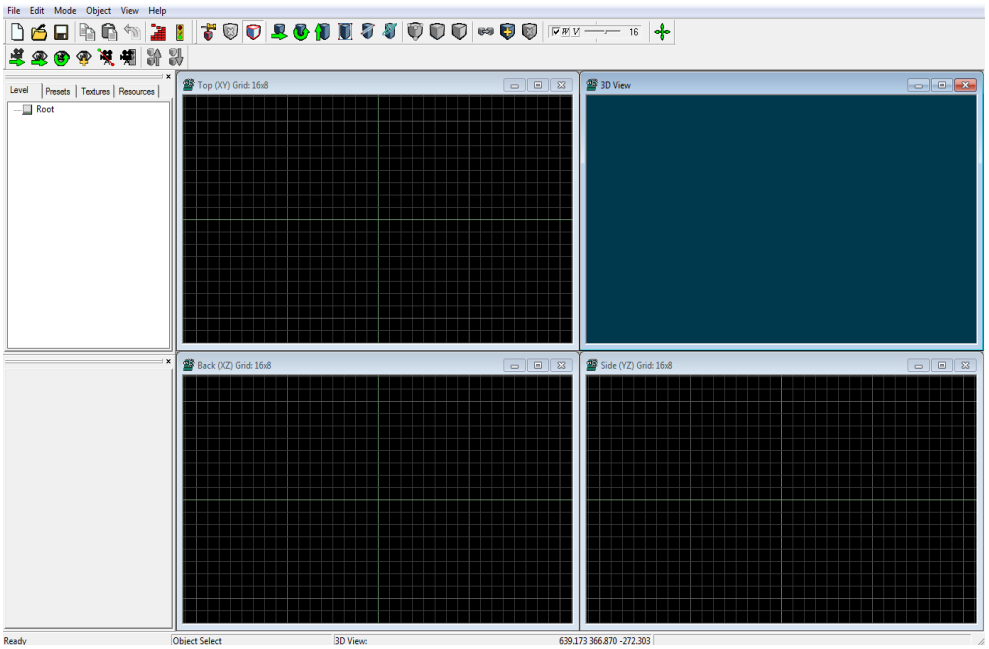

Figura 57: Captura WED.

En la siguiente tabla se explican brevemente las funciones de los botones que aparecen en la barra de herramientas:

	Crear nuevo proyecto wed.
	Abrir un proyecto wed existente.
	Guardar proyecto wed.
	Compilar el proyecto, siempre que se hace cualquier cambio hay que compilarlo para que wed lo incorpore al nivel y pueda hacer todos los cálculos necesarios para entre otras cosas, detectar colisiones entre objetos.
	Corre la aplicación y se puede ver el resultado final del mundo 3D creado.
	Mover un objeto. (Cuando se selecciona un objeto aparece el eje de coordenadas cartesianas y según el vector que se pinche el objeto se moverá en esa dirección).
	Rotar un objeto sobre el eje seleccionado.
	Modificar el tamaño de un objeto manteniendo sus proporciones.
	Permite moverse dentro de los planos para ver el nivel, además permite hacer zoom con la rueda del ratón.
	Parecido al anterior botón, pero no permite hacer zoom.
	Rota la vista con que se está viendo el plano.
	Hace zoom sobre la vista que se esté viendo.
	Añadir un objeto.
	Eliminar el objeto seleccionado.

Tabla 65: Barra de herramientas de WED.

De todos estos botones se explica en más detalle el botón que añade un objeto. Al pulsarlo aparece un listado que se muestra en la figura 58, que permite añadir desde objetos comunes y simples a luces, sonidos, cámaras, caminos que serviría para hacer que un personaje controlado por la máquina lo siguiera.

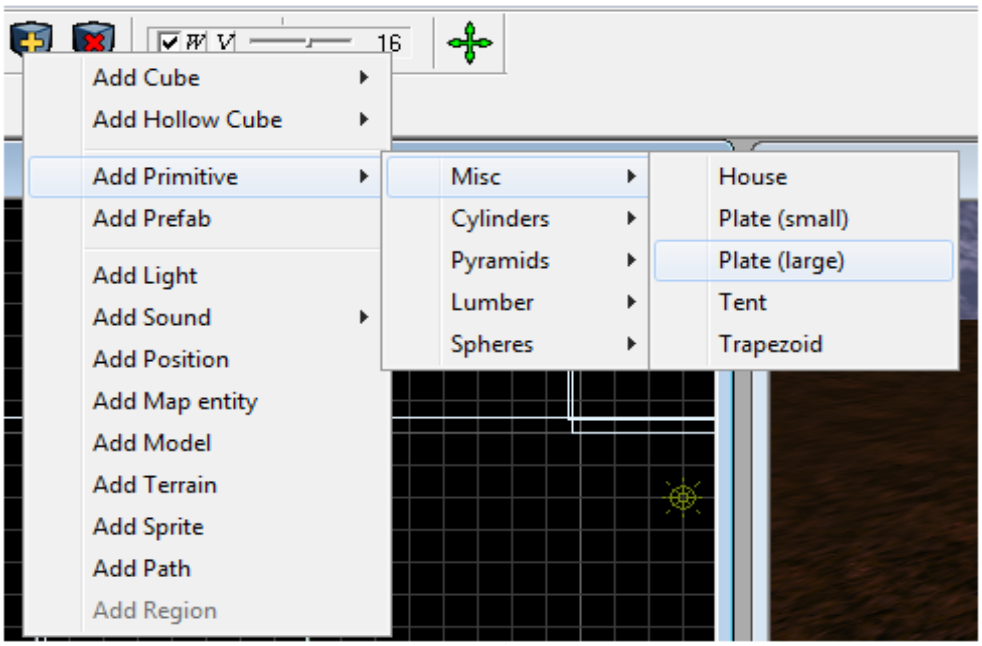

Figura 58: Botón añadir objeto en WED.

Por último, es conveniente saber cómo se aplica una textura a un objeto creado. Para ello basta con selecciona el objeto, y seleccionara en la ventana de la izquierda de la aplicación la pestaña "*Textures*", tal y como se puede ver en la figura 59. En esta pestaña se tiene acceso a las texturas por defecto que trae WED y se pueden importar otras nuevas. Una vez que se tiene la textura elegida, en las pestañas de abajo se elige "*Surface*" y se aplica a las caras que se deseen del objeto, dicha textura.

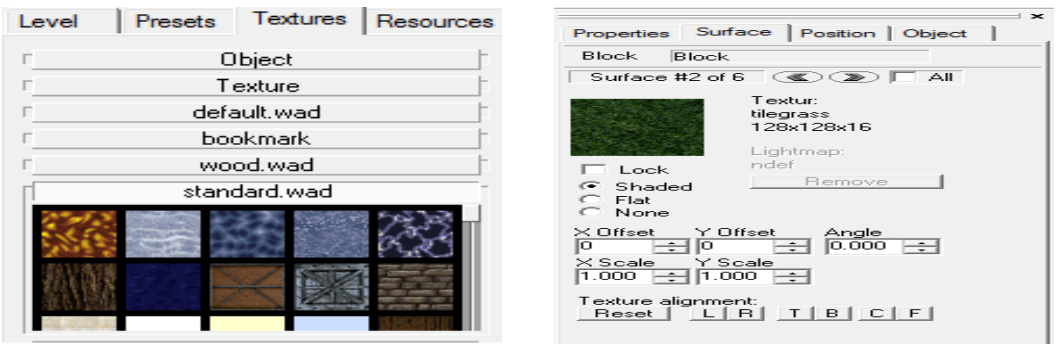

Figura 59: Modelo poligonal del vikingo.

Con estos conocimientos ya se puede implementar un nivel. Se parte del plano de la figura 59, donde la "*X*" representa muro o pared, la "*C*" representa una caja, la "*P*" representa al personaje y por último la "*D*", representa la zona en la que se tiene que dejar una caja.

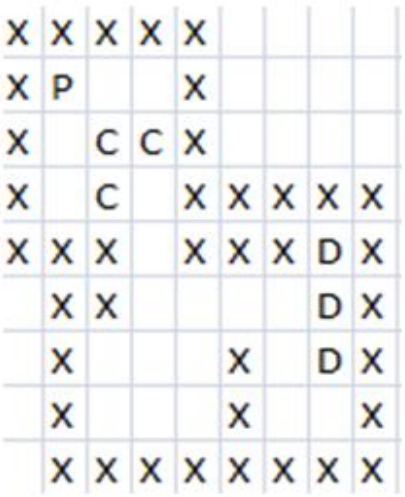

Figura 60: Plano nivel 1.

A continuación se enumeran los pasos seguidos para el nivel 1 aunque son válidos para cualquiera de los niveles creados:

1. Se crea un Hollow Cube, lo suficientemente grande para contener en su interior todo un nivel. Este objeto es un cubo hueco, que sirve para delimitar el nivel dentro de este.

2. Se aplica a todas las caras del cubo anterior la textura "*Skyblue*", que es una de las que trae por defecto wed. Esta textura hará que en vez de verse un fondo azul infinito en la aplicación, haga un efecto de cielo.

3. Se crea varios planos (Plate(large)) que unidos harán el suelo del nivel, con la forma exacta de este.

4. Se van añadiendo cubos y ajustándolos en tamaño, y se ponen encima del suelo del nivel para que hagan de muros.

5. Se da textura a todos los elementos del nivel.

6. Se añaden las luces por el escenario, estas son las que lo iluminarán cuando este corriendo la aplicación por lo que es importante que se ajusten adecuadamente. Un objeto del tipo luz es totalmente configurable. Se puede modificar desde su radio de acción, su intensidad e incluso el color de la luz que da.

En la figura 61 se puede ver cómo queda implementado el nivel 1 en la vista de arriba a la derecha, y en el resto de vistas se ven los diferentes planos del nivel.

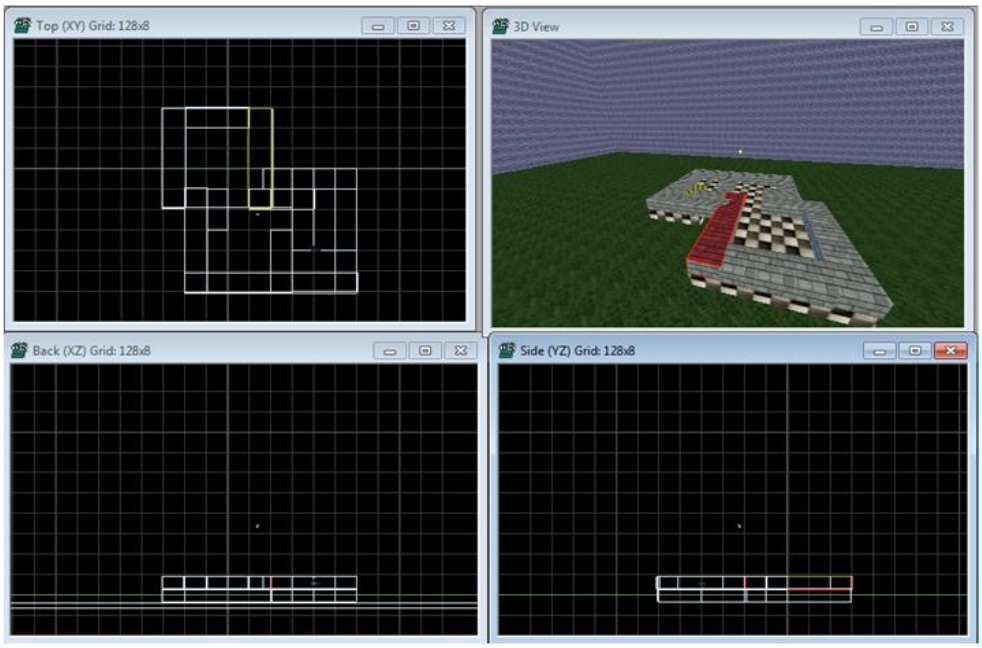

Figura 61: Nivel 1 hecho en WED.

En la figura 62 se ve una captura del nivel 1 desde arriba, aquí faltarían añadir las cajas y el personaje que se cargan en el nivel en el momento que se inicializa este. En este nivel habrá tres cajas en total.

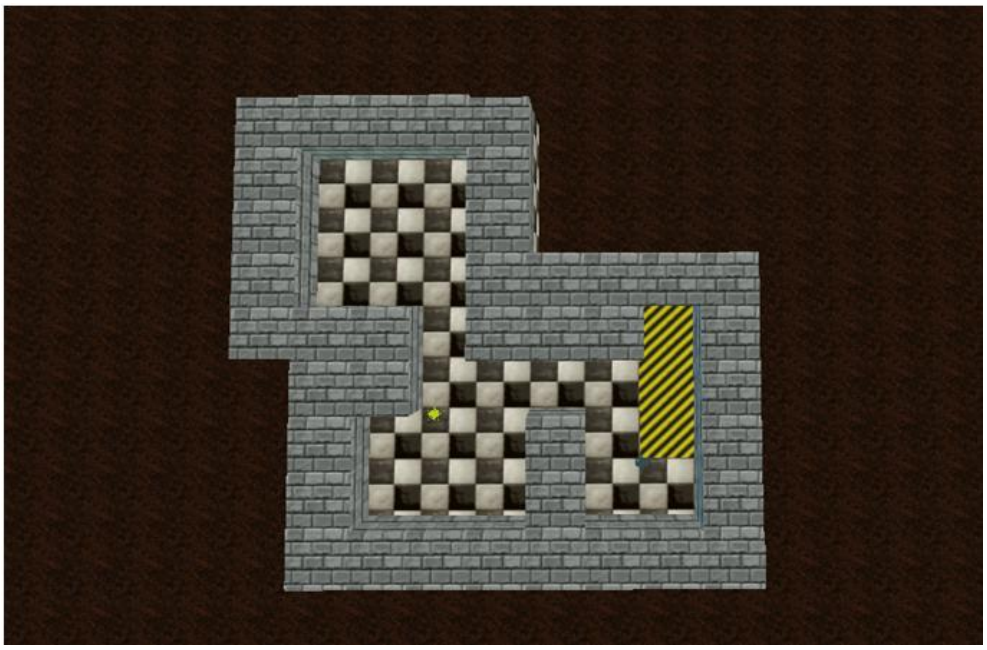

Figura 62: Resultado final del nivel 1.

En la figura 63 se ve una captura del nivel 2, este tendrá cuatro cajas a colocar.

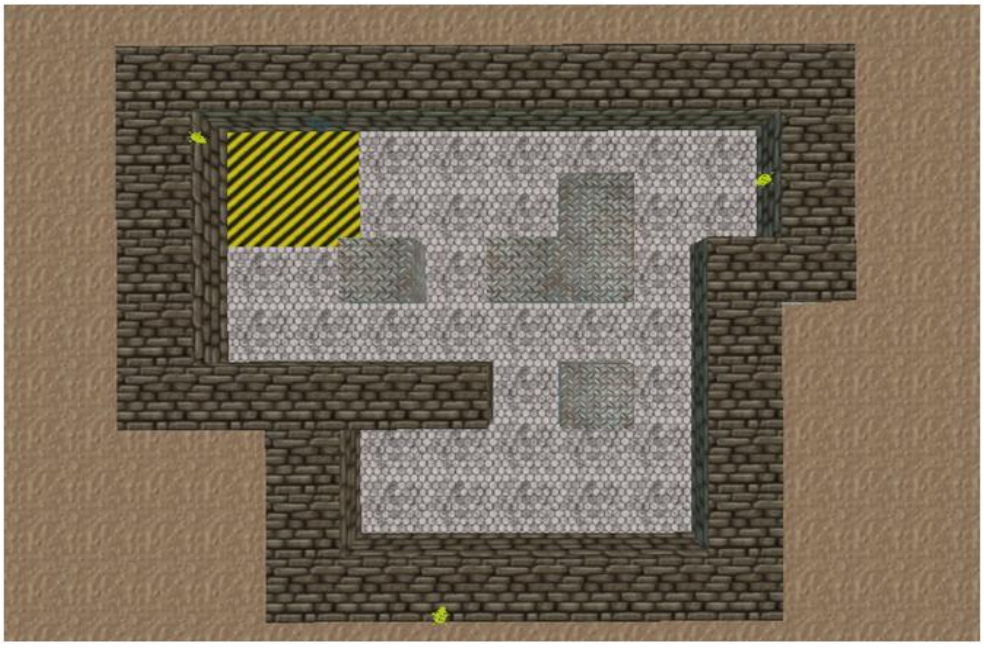

Figura 63: Resultado final del nivel 2.

Finalmente, en la figura 64 se ve una captura del nivel 3 finalizado, este tendrá 6 cajas.

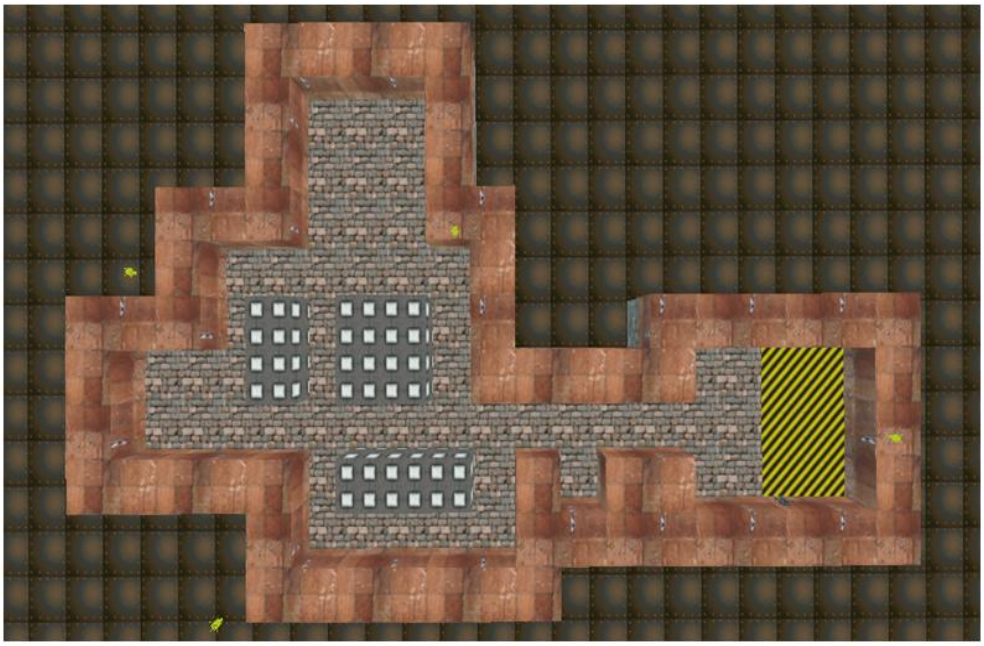

Figura 64: Resultado final del nivel 3.

3.3.3 SED (Script Editor)

Esta aplicación es un editor para crear, editar y depurar scripts. En la figura 65 se puede ver una captura de esta aplicación y en la tabla 66 los botones más relevantes de la barra de herramientas.

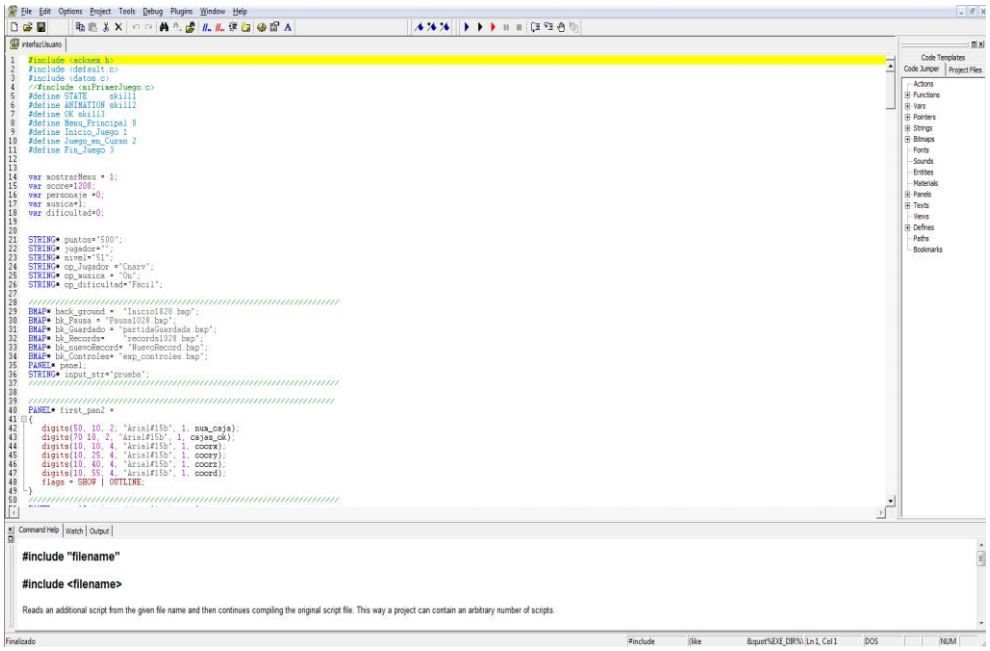

Figura 65: Captura SED.

	Crear nuevo script.
	Abrir un script existente.
	Guardar proyecto wed.
	Comentar y quitar comentarios de una línea de código.
	Correr o ejecutar la aplicación.
	Correr la aplicación en modo depuración de código.

Tabla 66: Barra de herramientas de SED.

3.3.4 Implementación del código

Una vez que se ha visto cómo se han implementado los personajes, las cajas y los niveles, se va a pasar a analizar la implementación del código. Como ya se había comentado al inicio de la explicación de esta fase, no se va a entrar a explicar todo el código sino lo más relevante de cada componente de la arquitectura de este software:

- Datos:
  - Apertura y cierre de ficheros.
  - Lectura y escritura de ficheros.
- Entrada/Salida:
  - Navegación de menús.
  - Entradas de periféricos tales como teclado y ratón.
  - Sonido.
  - Cámara
- Núcleo:
  - Control del personaje.
  - Control de las cajas y cómo se manejan las colisiones.
  - Control del tiempo.

DATOS

Manejo de ficheros

Como se había planteado en la fase de diseño, se ha creado un fichero llamado "*Data.CFG*". En este se guardan los datos de hasta tres partidas, que son las que va a permitir guardar la aplicación. También se guardan los datos de los cinco records de mayor puntuación con la puntuación y el nombre de quién los consiguió. La estructura del fichero esta divida en secciones y cada una de estas tiene varias claves. En la tabla 67 se muestra esta estructura.

[Partida1]	[RecordsP]	[RecordsN]
Nombre1 =	Jugador1 =	JugadorN1 =
Personaje1 =	Jugador2 =	JugadorN2 =
Puntos1 =	Jugador3 =	JugadorN3 =
Nivel1 =	Jugador4 =	JugadorN4 =
Dfctad1 =	Jugador5 =	JugadorN5 =

Tabla 67: Estructura del fichero Data.CFG.

Los nombres que están entre corchetes se corresponden con secciones y el resto con claves. De la sección "[*Partida1*]", se tienen las claves en las que se guardan todos los datos necesarios para poder continuar una partida guardada, que son el nombré con el que se guardó, el personaje con el que se jugaba, los puntos que llevaba, el nivel que estaba jugando y con qué dificultad. Como se guardan tres partidas, habrá tres secciones enumeradas de la 1 a la tres. Hay otras dos secciones más en las que las claves almacenan los nombres de los 5 records de mayor puntuación y las puntuaciones de dichos records.

Como también se comentó en la fase de diseño, este fichero no tiene ninguna seguridad definida ya que no era un requisito inicial de la aplicación aunque sí es una mejora necesaria para futuras versiones.

Para el control de este fichero, se ha creado un script, al que se le ha llamado "Datos.c". A continuación se analizan las funciones básicas para dicho control.

En el código 1, se ve como se hace la carga del archivo que se le pasa como parámetro. Abre el fichero "*Data.CFG*" y lo recorre línea por línea guardando toda la información en una variable global llamada "*_Profile*", también guarda en "*ini_nLinesC*" el número de líneas que tiene el fichero.

```
function Archivo_Cargar ( STRING* _sFile )
{
     var fhandle = 0;
     var c_asc = 0;
     STRING* c = "";
     var nLineas = 0;
     STRING* linea = "";
     fhandle = file_open_read ( _sFile );
     if ( fhandle )
     {
            while (1)
            {
                   while( c_asc != 10 )
                   {      c_asc = file_asc_read( fhandle );
                          if ( c_asc == -1 )
                          {
                                 file_close( fhandle ) ;
                                 ini_nLinesC = nLineas;
                                 return( nLineas );
                          }
                          if (c_asc != 13 && c_asc != 10)
                          {
                                 str_for_asc( c, c_asc );
                                 str_cat( linea, c) ;
                          }
                          else
                          {
                                 str_cat( linea, " ") ;
                          }

                   }
                   str_cpy( (_Profile.pstring)[nLineas], linea );

                   str_cpy( linea, "");
                   c_asc = 0;
                   nLineas ++;
            }

     }retun(0);
}
```

Código 1: Carga del fichero.

Una vez cargado el fichero, se ha implementado una función, "*Archivo_leer*" (se puede ver en código 2) que lo que hace es buscar dentro de la variable "*_Profile*", los valores con un sistema de búsqueda basado en la estructura de dicho fichero. De tal manera que esta función busca primero la sección y una vez localizada busca la clave y finalmente extrae el valor de la clave.

Así que para cargar todos los datos almacenados, lo que se hace primero, es cargar el fichero, a continuación buscar todos los valores de todas las claves llamando varias veces a la función anteriormente descrita, y guardar los resultados obtenidos en una serie de variables globales, a las que se puede acceder y modificar en cualquier momento. De esta manera se evita tener que acceder al fichero cada vez que se quiere consultar un dato guardado.

```
function Archivo_Leer ( STRING* _sSection, STRING* _sKey, STRING* _sValue )
{
	STRING* sNombreSeccion = "#256";
	STRING* sEntrada = "#256";

	var nNumLinea = 0;
	var bSectEncontrada = 0;
	var nPosIgual = 0;
	var nPosBlanco = 0;

	str_cpy( _sValue, "" );

	while( nNumLinea < ini_nLinesC )
	{
		if ( str_stri( (_Profile.pstring)[nNumLinea], "[") == 1 )
		{
			if (bSectEncontrada)
			return(0);

			str_cpy(sNombreSeccion, (_Profile.pstring)[nNumLinea]);
			str_clip(sNombreSeccion, 1);
			str_trunc(sNombreSeccion, 1);
			bSectEncontrada = (str_cmpni(_sSection, sNombreSeccion)
|| bSectEncontrada);
		}
		else
		{
			if (bSectEncontrada)
			{
				nPosIgual =
str_stri((_Profile.pstring)[nNumLinea], "=");
				if (nPosIgual)
				{
					str_cpy(sEntrada,
(_Profile.pstring)[nNumLinea]);
					str_cpy(_sValue, sEntrada);
					str_trunc(sEntrada, (str_len(sEntrada) -
nPosIgual) + 1);
					if (str_cmpni(_sKey, sEntrada))
					{
						str_clip(_sValue, nPosIgual);

						// ELIMINAMOS LOS ESPACIOS EN BLANCO
ENTRE EL IGUAL Y EL VALOR QUE QUEREMOS OBTENER
						while ( str_stri(_sValue, " ") == 1 )
						{
							str_clip(_sValue, 1);
						}

						return(nNumLinea);
					}
				}
			}
		}

		nNumLinea ++;
	}

	return(0);
}
```

Código 2: Lectura del fichero.

Para modificar el fichero se hace un proceso parecido al de la lectura puesto que se lee de nuevo la variable "*_Profile*" buscando el número de línea del valor a modificar y una vez encontrado se sobrescribe, tal y como se ve en el código 3.

```
function Archivo_Modificar ( STRING* _sSection, STRING* _sKey, STRING*
_sValue )
{
     var nNumLinea = 0;
     STRING* sEntrada = "#20";
     STRING* last_value = "1";
//   STRING* enter= "";
//   str_for_asc( enter, 13 );
     nNumLinea = Archivo_Leer( _sSection, _sKey, last_value );
     if (nNumLinea)
     {
            str_cpy(sEntrada, _sKey);
            str_cat(sEntrada, " = ");
            str_cat(sEntrada, _sValue);
//          str_cat(sEntrada, enter);

            str_cpy( (_Profile.pstring)[nNumLinea], sEntrada);
            return nNumLinea;
     }

     return 0;
}
```

Código 3: Modificación del fichero.

Una vez que se ha sobrescrito el valor, se llama a la función "*Archivo_Guardar*", que lo que hace es recorrer la variable "*_Profile*" línea por línea, y copiándolas en el fichero "*Data.CFG*", finalmente cierra el fichero con los datos cambiados.

```
function Archivo_Guardar( STRING* _sFile )
{
     var fhandle = 0;
     var nLineas = 0;
     STRING* linea = "#40";
     var aa = 0;

     fhandle = file_open_write ( _sFile );

     if ( fhandle )
     {
            while ( nLineas < ini_nLinesC )
            {
                   str_cpy ( linea, *(_Profile.pstring)[nLineas] );

                   file_str_write ( fhandle, linea );
                   file_asc_write ( fhandle, 10 ); // Salto de línea
                   nLineas ++;
            }
            file_close( fhandle );
            return(1);
     }

     return(0);
}
```

Código 4: Guardar el fichero.

Para el guardado de partidas y de records se han implementado otras funciones que llaman a las que se acaban de analizar, pasándoles los parámetros a modificar y guardar.

ENTRADA/SALIDA

Navegación de menús

Para la presentación en pantalla de los menús y textos, se ha basado toda la implementación en el uso de dos tipos de variables que son PANEL y TEXT.

Un panel es una imagen rectangular que se pone encima de la pantalla, de tal manera que tapa parte del mundo 3D generado por el motor gráfico. Se usan para hacer menús que deben ocupar toda la pantalla, o paneles que aparezcan mientras se está jugando y den información del tiempo o la vida que le queda al personaje o la puntuación obtenida. En la figura 66 se ven un par de ejemplos, en la imagen de la izquierda se muestra un panel en el que podrían mostrarse los objetos que tiene un personaje y el fondo azul sería el mundo 3D del juego. En la imagen de la derecha, se ve una captura del resultado final del proyecto, en la parte superior se ve el panel del juego que indica la puntuación que lleva el jugador, el número de cajas colocadas y el tiempo que le queda para superar el nivel.

Figura 66: Ejemplos de panel.

La variable panel tiene una serie de atributos totalmente configurables para su aparición en pantalla. Uno de ellos es "*Layer*", este es un valor numérico que le asigna una prioridad a este panel, de tal manera que si se muestran dos paneles y en alguna parte de la pantalla se superpone uno encima de otro, el que se acabará mostrando es el que tiene un valor mayor de layer. Otro atributo que tiene, es un flag o bandera, que indica si este panel es visible o no, de tal manera que simplemente modificando este valor se hará que se oculte o que se muestre dicho panel.

Se han utilizado paneles para hacer todos los menús del juego, el proceso es bastante sencillo, y se va analizar con la implementación del menú de pausa y servirá de referencia para el resto de menús.

Lo primero que se ha hecho es crear con el paint una imagen que servirá de fondo de pantalla para dicho menú. Esta imagen se puede ver en la figura 67. Como se puede observar se ha cumplido con una de las pautas marcadas en la fase del diseño, que era que los menús aparecieran imágenes de cajas.

Figura 67: Imagen menú pausa.

Posteriormente se han creado los botones que irán superpuestos encima de la caja con el paint o con eezPix. Para crear un efecto diferente cuando el ratón se pone sobre el botón, se vuelve a hacer el mismo botón pero con otros colores, tal y como se ve muestra en la figura 68.

Figura 68: Botones continuar.

Una vez que ya se han creado todos las imágenes a mostrar ya se puede crear el panel tal y como se muestra en el código 5. En la definición del panel se ha indicado en qué posición se va a colocar, el número se refiere al número de pixel de la pantalla teniendo en cuenta que el eje

horizontal sería la x y el vertical la y, y que las coordenadas del origen estarían en el primer pixel de la esquina izquierda superior de la pantalla. Se le ha asignado un valor de layer bajo porque es la primera capa que se va a mostrar y el resto irá encima de este. En las siguientes líneas se añaden los botones con las funciones que va a tener este menú, cada botón indica qué imagen se muestra de inicio, cuál se muestra cuando se coloca el puntero del ratón encima, y finalmente la función a la que llama cuando se hace click sobre este. Y por último se configura el flag, comentado anteriormente, como "*Show*". Con lo que el panel en el momento de su definición estará visible hasta que no se modifique el flag o se ponga un panel con un layer mayor encima.

```
PANEL* panelPausa =
{
     pos_x = 0;
     pos_y = 0;
     layer = 1;
     bmap = bk_Pausa;
     button (480, 356, "reiniciar.pcx", "reiniciar.pcx",
"reiniciar_over.pcx", reiniciarPartida,NULL,NULL);
     button (480, 426, "guardar.pcx", "guardar.pcx","guardar_over.pcx",
mostrarGuardar,NULL,NULL);
     button (480, 496, "continuar.pcx",
"continuar.pcx","continuar_over.pcx", continuarPartida,NULL,NULL);
     button (480, 566, "salir.pcx", "salir.pcx","salir_over.pcx",
salirPartida,NULL,NULL);
     flags = SHOW;
}
```

Código 5: Código panel de pausa.

A continuación se muestran las instrucciones para modificar este flag, estas se pueden usar en cualquier momento de la aplicación ya que los paneles se han definido como variables globales.

```
set(panelPausa,SHOW);// Pone el flag de este panel como visible.

reset(panelPausa,SHOW);// Pone el flag de este panel como no visible.
```

Código 6: Instrucciones para modificar flag de un panel.

Con estos sencillos pasos se ha implementado el menú de pausa, el resultado final se puede observar en la imagen 69. Para el resto de menús se ha seguido la misma metodología, y la aplicación llama a diferentes funciones que ejecutan las instrucciones mostradas en el código 6, para ir mostrando y ocultando los menús según lo requiera el juego en ese momento.

Figura 69: Menú de pausa finalizado.

Las variables del tipo TEXT son similares a las de panel, lo único en que varían es que en vez de mostrar por pantalla una imagen, muestran un texto. Estas se han usado como complemento de los paneles. Por ejemplo, en el panel que se comentó anteriormente, en el que se muestran el tiempo, las cajas colocadas y la puntuación, los números que aparecen son variables tipo Text. Así que la imagen que no varía se implementa con un panel y los textos y dígitos que pueden tener diferentes valores se implementan con Text. En el código 7 se ve cómo se define este tipo de variables, indicando su posición en pantalla, sus colores, su valor de layer, este tiene que ser superior al del panel para que se vea el texto.

```
TEXT* opciones1_txt =
{
  layer = 2;
  pos_x = 690;
  pos_y = 370;
  flags = SHOW;
  red=255;
  green=0;
  blue=0;
  font = "Arial#25b";
}
```

Código 7: Definición de variable tipo TEXT.

En el código 8 muestra cómo modificar el valor de estas variables y por tanto, el mensaje que aparece en pantalla.

```
str_cpy((guardar_txt.pstring)[0], string2);
```

Código 8: Modificación de variable TEXT.

Uso del ratón

Para el control del ratón, el motor gráfico maneja una serie de variables predefinidas, entre estas está "*mouse_mode*". En función del valor que se le asigne a esta, se podrá a llegar a habilitar o deshabilitar este periférico. Se le puede asignar los siguientes valores:

- "0", el puntero del ratón se hace invisible y no tiene ninguna utilidad, es decir se inhabilita.
- "1", el puntero de ratón es visible y además controla las cámaras, útil para juegos en primera y tercera persona.
- "2", el puntero de ratón es visible pero no controla las cámaras, útil en juegos de estrategia.
- "4", el puntero del ratón es visible y sigue automáticamente al ratón de Windows.

En los requisitos se planteo que el juego se controlaría con el teclado y el ratón sólo se usaría para navegar por los menús. Con lo que mientras se está jugando la partida se asigna a la variable "*mouse_mode*" el valor 0 de tal manera que queda inhabilitado y cuando se sale de este modo de juego, partida se asigna a la variable "*mouse_mode*" el valor 4.

También se pueden asociar funciones al suceso de eventos como la pulsación de un botón del ratón, aunque no se han llegado a usar para la aplicación.

Uso de teclado

En cuanto al uso del teclado, este no se habilita o deshabilita como el ratón sino que siempre está operativo. Entre las funciones predefinidas de las que dispone este motor, se encuentra el poder asociar un evento determinado a la pulsación de cualquier tecla, es decir, que si se pulsa una determinada tecla, hace que se ejecute otra función en ese mismo momento. El código 9 muestra dos instrucciones usadas en la aplicación.

```
on_space = posCamara;
on_p = mostrarPausa;
```

Código 9: Eventos de teclado.

Estas líneas de código se han implementado en la función main o principal, de tal manera que en el momento que se pulse la tecla espacio, se ejecute a continuación la función

"*posCamara*", de la que más adelante se verá su uso. Si por el contrario, se pulsa la tecla p, se ejecutará la función "*mostrarPausa*". Esta función comprobará si se está jugando una partida en ese momento y de ser así, la parará y mostrará el menú de pausa.

En cuanto al segundo uso del teclado que se quería destacar, era la introducción de datos desde este y que aparezcan reflejados por pantalla. Un ejemplo se puede ver en la función "*escribirRanura1*" en el código 10. Esta función se usa para introducir el nombre con el que el usuario quiera guardar la partida en la posición 1. Lo primero que se hace es inhabilitar el ratón y a continuación meter en un bucle la aplicación, y se va modificando la cadena de texto de la variable guardar_txt, hasta que el caracter introducido es una pulsación de enter. Cuando sucede esto, sale del bucle y habilita el ratón de nuevo. La instrucción "*inchar(string1)*", hace que el programa espere a una pulsación de teclado, cuando se produce introduce en string1 la tecla pulsada. También es importante la instrucción "*key_lastpressed*", esta es una variable que almacena automáticamente el valor en hexadecimal de la última tecla presionada.

```
function escribirRanura1()
{
	STRING* string1 = "";
	STRING* string2 = "";
	mouse_mode =0;
	while(1)
	{
		if (inchar(string1) == 13) break;

		if(key_lastpressed==14)
		{
		 str_trunc(string2,1);
		}
		else
		{
			if(str_len(string2)<21) str_cat(string2,string1);
		}
		str_cpy((guardar_txt.pstring)[0], string2);

	}
	str_cpy(string2,"");
	mouse_mode=4;


}
```

Código 10: Escribir en pantalla.

Control de sonido

Todos los sonidos usados en esta aplicación se han obtenido de manera gratuita de internet {93}. Para el control del sonido primeramente se definen las variables del tipo SOUND, que tienen el path de el archivo de sonido con extensión .wad, tal y como se muestra en el código 11.

```
SOUND* SND_MENU = "menu.wav";
SOUND* SND_JUEGO = "juego.wav";
SOUND* SND_GAME_OVER = "gameover.wav";
SOUND* SND_SCORE = "score.wav";
SOUND* SND_CAJA = "caja_ok.wav";
SOUND* SND_BONUS = "bonus.wav";
```

Código 11: Definición de variables de sonido.

También se han definido dos variables globales que sirven para el control del sonido en todo momento.

```
var musica=1;
var punteroMusica = 0;
```

Código 12: Definición de variables para el control del sonido.

Con la primera se controla si el usuario quiere que haya sonido o no, de tal manera que cuando desde el menú de opciones se elige que no se quiere sonido, este valor se pone a 0 y parará toda reproducción de sonido.

Tal y como se ve en el ejemplo del código 13, para reproducir un sonido, en este caso de manera cíclica (cuando acaba el sonido, vuelve a empezar), se utiliza esta instrucción. Como parámetros se le pasa la variable del sonido a reproducir y el volumen del sonido. Esta devuelve un valor numérico que se recoge en la variable "*punteroMusica*".

```
punteroMusica = snd_loop(SND_MENU,30,0);
```

Código 13: Ejemplo de reproducción de un sonido.

Para parar un sonido hay que utilizar la función que se muestra en el código 14, y para que identifique qué sonido debe parar, hay que pasarle cómo parámetro el valor numérico que devolvía la función que inicio su reproducción. De ahí que previamente se haya guardado este valor en "*punteroMusica*", de no ser así no se podría parar un sonido que se ha reproducido de manera cíclica.

```
snd_stop(punteroMusica);
```

Código 14: Parar sonido que se está reproduciendo.

En el código 15 se muestra la función encargada de controlar el sonido de la música de fondo. Lo primero que hace es verificar si está habilitada la música, si no es así, no reproduce ningún sonido. Por el contrario, si la música está habilitada sí va a reproducir un sonido, este variará en función de si se está navegando por los menús o se está jugando la partida.

```
function controlSonido(var sonido)
{
      if(musica)
      {
            if(sonido == 0)
            {
                  wait(10);
                  snd_stop(punteroMusica);
                  punteroMusica = snd_loop(SND_MENU,30,0);
            }
            else
            {
                  wait(10);
                  snd_stop(punteroMusica);
                  punteroMusica = snd_loop(SND_JUEGO,30,0);

            }
      }

}
```

Código 15: Función control del sonido.

Además de la música de fondo se han añadido más sonidos para diferentes eventos, tales como cuando se coloca una caja correctamente, se establece un nuevo record o se acaba el tiempo del nivel

Cámara

Como ya se ha comentado anteriormente se han implementado tres ángulos de vistas. La primera vista es en tercera persona, la cámara se sitúa por detrás del personaje y ligeramente más alta que este, tal y como se puede ver en la figura 70.

Figura 70: Vista tercera persona.

La siguiente vista se corresponde con un plano aéreo del nivel, este plano se mueve con el movimiento del personaje, tal y como se muestra en la figura 71.

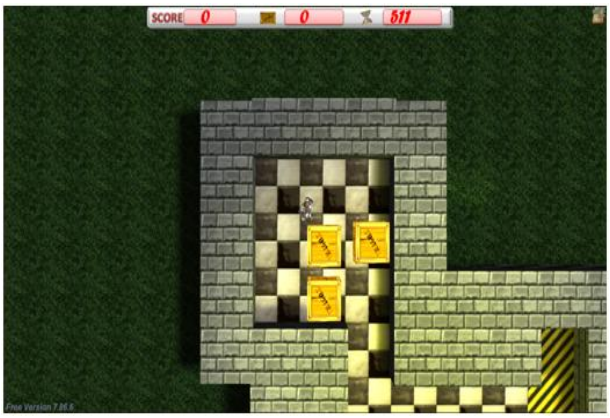

Figura 71: Vista desde arriba.

Y la última disponible, también se corresponde con un plano aéreo, pero en este caso es más lejano y permite ver el nivel por completo, tal y como muestra la figura 72.

Figura 72: Vista desde arriba de todo el nivel.

El motor 3D Game Studio tiene una variable predefinida llamada "*camera*", con una serie de atributos que permiten configurar lo que va a captar dicha cámara. De estos atributos los más importantes son los que tienen que ver con la posición de la cámara y los que tienen que ver con el ángulo de la cámara. Es decir, por un lado se indica dónde se coloca la cámara en el nivel y por otro lado hacia dónde mira. En el código 16 se muestran los atributos que tienen que ver con la posición de la cámara en el nivel y se muestra un ejemplo de cómo indicar que la cámara se encuentra en las coordenadas (-200, 5, 150).

```
camera.x
camera.y
camera.z
....

vec_set(camera.x,vector(-200,5,150));
```

Código 16: Atributos de posición de la cámara.

En cuanto a los atributos que tienen que ver con el ángulo de la cámara, indicar que son tres:

- Pan, es el ángulo de giro de la cámara sobre en el eje horizontal. Sus valores están comprendidos entre 0 y 360 grados.

- Tilt, es el ángulo de giro de la cámara en el eje vertical. Sus valores están comprendidos entre -90 y 90 grados.

- Roll, es el ángulo de giro de la cámara sobre sí misma. Sus valores están comprendidos entre 0 y 360 grados.

En la figura 73 se muestra gráficamente el valor de estos ángulos.

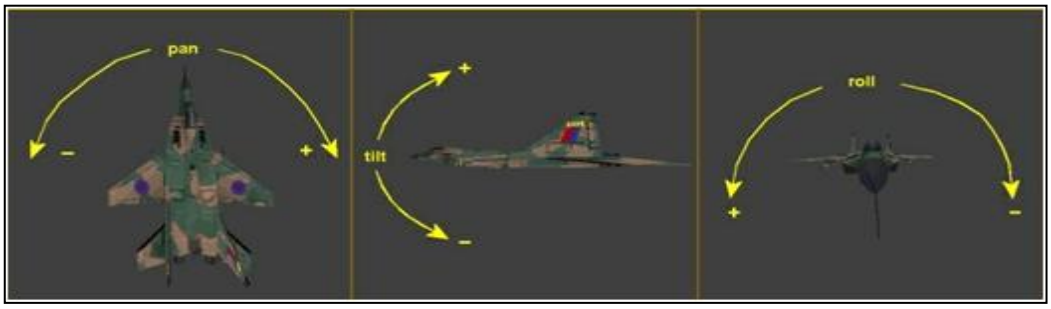

Figura 73: Ángulos pan, tilt y roll.

Para el control de la cámara en el juego, primeramente se ha definido una variable global llamada "*camara*", que sus valores van de 0 a 2, cada uno se corresponde con una vista del juego. Como ya se explico en el uso del teclado, cuando el usuario pulsa la tecla espacio, se ejecuta la función "*posCamara*", esta lo que hace es modificar el valor de la variable camara.

Por otro lado, se ha implementado una función que por parámetro se pasa la entidad del personaje. Esta función ajusta los valores de la variable predefinida "*camera*", en función de la vista seleccionada. Es necesario que se pase la entidad del jugador como parámetro, ya que para poder ajustar la cámara al jugador, es necesario conocer la posición de este. También,

como se tienen dos personajes diferentes, con diferentes tamaños, se varían la posición de la cámara en función del personaje elegido. El código completo se puede ver en código 17.

```
function posicion_camara(ENTITY* ent)
{
	while(1)
	{

	if(camara==0)
		{
			if(personaje==0)
			{
				vec_set(camera.x,vector(-200,5,150));  //
posiciona la cámara con respecto a la entidad
				vec_rotate(camera.x,ent.pan); // rota la cámara
con la posición respecto al jugador
				vec_add(camera.x,ent.x);	// add player
position
				vec_set(camera.pan,vector(ent.pan,-30,0)); // look
in player direction, slighty down
			}
			else
			{
				vec_set(camera.x,vector(-300,5,150));  //
posiciona la cámara con respecto a la entidad
				vec_rotate(camera.x,ent.pan); // rota la cámara
con la posicion respecto al jugador
				vec_add(camera.x,ent.x);	// añade la posición
de la entidad
				vec_set(camera.pan,vector(ent.pan,-30,0)); // mira
en la dirección del jugador
			}
		}
	else
		{
			if(camara==1)
			{
			vec_set(camera.x, vector (ent.x,ent.y,1500));
			vec_set(camera.pan,vector(90,-90,0));
			}
			else
			{
			vec_set(camera.x, vector (0,0,3000));
			vec_set(camera.pan,vector(90,-90,0));
			}

		}
	wait(1);
	}
}
```

Código 17: Función del control de la cámara.

NÚCLEO

Control del personaje

Para crear el personaje se han usado entidades, la entidad es un objeto dinámico que se puede mover por el escenario. A estas se le pueden asignar una acción, que es una función que se ejecuta en el momento que se crea la entidad y es un proceso paralelo al principal. En el caso de un jugador que se va a mover por el escenario se programa dentro de esta acción un bucle que detecta las pulsaciones de teclado.

En el caso de esta aplicación, la entidad del jugador se crea en el momento que se inicializa el nivel y se destruye cuando se acaba el nivel. A la entidad jugador se le asigna la acción "*jugador_andando()*" tal y como se puede ver en el código 18.

```
ENTITY* jugador;  // Definición de la entidad, junto con el resto de
                  //variables globales

....

jugador = ent_create ("vikingo.mdl", vector(-192,176,85), jugador_andando);
            // Inicialización de la entidad asignándole el personaje del
            //vikingo, indicando la posición dónde se coloca y asignándole
            //la acción de jugador_andando
....

action jugador_andando()
{
     posicion_camara(my);//Llama a la función que controla la cámara
     while (1)
     {
     c_trace (my.x,vector(my.x,my.y,my.z-
1000),IGNORE_ME|IGNORE_PASSABLE|IGNORE_PUSH);
     my.pan += (key_cul-key_cur)*8*time_step;
     var distance = (key_cuu-key_cud)*8*time_step;
     c_move(my, vector(distance,0,0), NULL, GLIDE);
     my.ANIMATION += 2*distance;
     ent_animate(me,"walk",my.ANIMATION,ANM_CYCLE);
     c_trace(my.x,vector(my.x,my.y,my.z-2000),IGNORE_ME | IGNORE_PASSABLE |
IGNORE_PUSH);// Ajusta la posición de la entidad al suelo
     wait(1);
     }
}
```

Código 18: Control del jugador.

Para mover una entidad por el nivel se podría ir cambiando los valores de su posición cada vez que el usuario pulsara una determinada tecla, el problema que tiene esto es que el motor de colisiones no funcionaría en este caso, de tal manera que el personaje al encontrarse con una pared la atravesaría. Como esto no es algo contemplado en la aplicación, se opta por usar la instrucción "*c_move*", a esta se le indica que entidad tiene que mover y en qué dirección.

Como al compilar el nivel el motor gráfico ya ha generado el mapa de objetos, este detectará cuando el personaje choca con un obstáculo y por tanto no lo atravesará sino que se parará.

El jugador se mueve con las flechas del teclado, de tal manera que calcula la distancia que tiene que andar cada vez que se pulsa una de estas teclas. El uso la variable "*time_step*" es fundamental, ya que se utiliza para que independientemente de las características del ordenador donde se corra la aplicación, el jugador tarde lo mismo en hacer el mismo movimiento. Sino en ordenadores muy potentes que ejecuten las instrucciones del programa mucho más rápido, haría que el personaje se desplazará demasiado deprisa por el nivel.

Como en lite-c no se usan constructores, ni clases, no se pueden definir objetos con diferentes atributos como en otros lenguajes de programación. Pero se pueden usar skilles, que son atributos que se definen en el inicio del script, tal y como se muestra en el código 19. De esta manera, cada entidad podrá cambiar el valor de esos atributos propios, accediendo con su nombre seguido de un punto y el skill a modificar, como se ve en el código 21.

```
#define STATE     skill1
#define ANIMATION skill2
#define OK skill3
```

Código 19: Animación de una entidad.

Para que cuando se mueva el jugador por la pantalla haga el movimiento de andar tal y como se definió en la parte del MED, se utilizan dos instrucciones que aparecen en el código 19 pero que se vuelven a mostrar en el código 20.

```
my.ANIMATION += 2*distance;
ent_animate(me,"walk",my.ANIMATION,ANM_CYCLE);
```

Código 20: Animación de una entidad.

En la primera línea se asigna al skill "*ANIMATION*" de la entidad, el número del porcentaje del ciclo de animación a mostrar. La segunda línea ejecuta la animación de andar de la entidad que se le pasa por parámetro. Si el ciclo de animación de andar tiene en total 6 fotogramas, si es valor de animation es un cero se correspondería con el fotograma 1 y si fuera un 50% sería el fotograma 3, de tal manera que la animación mostraría los fotograma 1,2 y 3.

Control de las cajas

Para el manejo de las cajas, se ha creado una entidad por cada caja. Y se han definido dos skilles más. El primer skill llamado "*OK*", se utiliza para indicar que la caja esta correctamente colocada.

```
#define OK skill3
#define STATE     skill1
```

Código 21: Atributos de la entidad caja.

Este se puede modificar en cualquier momento de la partida accediendo a él con el nombre de la entidad seguido de un punto y el nombre del atributo, como por ejemplo:

```
caja1.OK = 1;
```

Código 22: Ejemplo de skill.

Así que cada vez que se coloque una caja, se modificará el valor de ok de esa entidad y se aumentara el contador cajas_ok, que lleva la cuenta de todas las cajas colocadas, y a su vez se verificará si el resto de entidades caja ya se han colocado, de esta manera la aplicación sabrá que se ha superado el nivel tal y como muestra el código 23.

```
function fin_juego()
{
     //Comprueba si para el nivel que se está jugando en ese momento
     var numNivel = str_to_num(nivel);
     var esFinal = 0;

     switch (numNivel)
     {
            case 1:
            if (cajas_ok == 3) esFinal = 1;
            break;

            case 2:
            if (cajas_ok == 4) esFinal = 1;
            break;

            case 3:
            if (cajas_ok == 6) esFinal = 1;
            break;
     }

     if(esFinal)
     {
            calcularPuntuacion();
            snd_stop(punteroMusica);
            wait(2);
            punteroMusica = snd_loop(SND_BONUS,30,0);
            esRecord();
     }
}
```

Código 23: Comprueba si se ha superado un nivel.

El skill STATE se utiliza para hacer una máquina de estados de la entidad. En el caso de una caja va a ser muy simple porque sólamente va a tener dos, o está quieta o la están empujando.

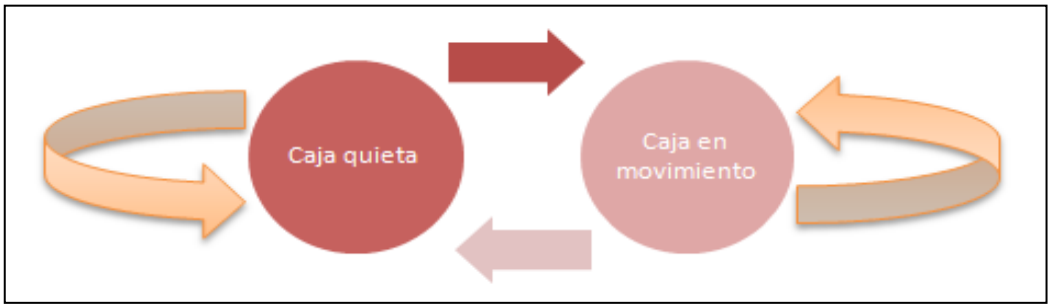

Figura 74: Diagrama de estados de la entidad caja.

Al inicializar el nivel se crean todas las entidades caja y a cada una se la asigna la acción caja_quieta, mostrada en el código 24.

Lo primero que se hace al implementar la acción de la caja es definir su bounding bones, es decir, se define el cuerpo de la caja susceptible de colisionar con otros objetos. Esto se puede ajustar a que sea más grande o más pequeño que la propia caja. Esto hará que el jugador puede llegar a tocar la caja estando lejos de ella o viceversa, que para que el motor de colisiones detecte que hay contacto entre el jugador y la caja, el primero tiene que atravesarla prácticamente.

A continuación se habilita el evento IMPACT, esto lo que hace es que cuando el motor de colisiones detecta que otra entidad ha impactado con la caja se ejecute una función, en este caso la función "*mover_caja*".

La acción entra en un bucle de estado 1, en el que está quieta y no hace nada hasta que no cambie el valor del skill STATE.

Cuando otra entidad colisiona con la caja, la función mover_caja modifica el valor del estado y hace que la acción salga de ese bucle. Como una caja no debe empujar otra caja, ni un jugador puede empujar dos cajas de tal manera que la primera empuje a la siguiente, lo que hace la acción es verificar qué tipo de entidad la está tocando. Para ello ejecuta la instrucción "*c_trace*", que lo que hace es lanzar un rayo en todas las direcciones y verificar con que es lo que ha colisionado primero.

Si es el jugador lo que la está empujando, hace una serie de cálculos para determinar desde que lado la están empujando para moverse en esa dirección.

Finalmente cuando deja de detectar que el jugador la empuja, verifica si está bien colocada llamando a la función "*caja_colocada*". En caso afirmativo, verifica si se ha superado el nivel llamando a la función "*fin_juego*", si por el contrario, no está bien colocada, vuelve a su estado inicial a la espera de que la vuelvan a mover.

Se podría haber hecho que la caja se moviera al empujarla el jugador de una manera más sencilla, habiéndola aplicado propiedades físicas tales como peso, rozamiento. El motor de colisiones gestionaría automáticamente que cuando el jugador tocara la caja esta se movería. Aun siendo un camino a priori más fácil, no se ha llegado a usar dada la dinámica del juego ya que las cajas no se mueven como se moverían en la realidad, ya que no se pueden mover en todas las direcciones, no deberías poder mover dos cajas en fila a la vez.

```
caja1 = ent_create ("crate.mdl", vector(64,64, 85), caja_quieta);
....

action caja_quieta()
{
     //*********Configuración bounding bones caja*************
     my.eflags |= FAT | NARROW; // set both flags to prevent automatic
recalculation on scale changes
     vec_set(my.min_x,vector(-30,-50,-45)); // set bounding box to
individual values
     vec_set(my.max_x,vector(30,50,45));
     //*********Fin Configuración *************
     my.emask |= ENABLE_IMPACT;
     my.event = mover_caja;
     my.STATE = 1;
     my.OK=0;


     while(1)
     {

     if(my.STATE==1)
     {
            // state 1: La entidad caja no hace nada, solo esperar a que la
muevan////////////////
            wait(1);
     }else
     {

            if(my.STATE==2)
            {

            // state 2: Otra entidad ha entrado en contacto con la
caja////////////////////////////////////

                  c_trace
(my.x,jugador.x,IGNORE_ME|IGNORE_PASSABLE|IGNORE_PUSH); // Lanzo rayo para
conocer el entorno que rodea a la caja
                  if(hit.entity==jugador) //Verifico si lo que está tocando
a la caja es el personaje
                  {
                        var i=0;
                        var distx= abs(my.x-jugador.x);//Calculo las
distancias de la caja al personaje en ambos ejes
                        var disty= abs(my.y-jugador.y);
```

```
                        while(i<2)
                        {
                                var signo=1;
                                if(distx>disty)//Compruebo por dónde está
empujando el personaje la caja para determinar en qué dirección mover la
caja
                                {
                                        if(jugador.x>my.x) signo =-1;
                                        var distance = signo*(key_cuu-
key_cud)*8*time_step;
                                        c_move(my, vector(distance,0,0),
NULL, GLIDE);
                                        i++;
                                }
                                else
                                {
                                        if(jugador.y>my.y) signo =-1;
                                        var distance = signo*(key_cuu-
key_cud)*8*time_step;
                                        c_move(my, vector(0,distance,0),
NULL, GLIDE);
                                        i++;
                                }
                        }
                        if(caja_colocada(my.x,my.y))
                        {
                                if(my.OK!=1)
                                {
                                        my.OK=1;
                                        cajas_ok++;
                                        snd_play(SND_CAJA,100,0);
                                        fin_juego();
                                }

                        }
                        else
                        {
                                if(my.OK)
                                {
                                        my.OK=0;
                                        cajas_ok--;
                                }
                        }
                }
                my.STATE=1;

        }
    }
    wait(1);
}
```

Código 24: Función del control de la entidad caja.

Control del tiempo

Finalmente se va a analizar cómo la aplicación controla el tiempo de juego, para ello se crea una entidad a la que se le ha llamado reloj, con una acción asignada que es contador. Al igual que el resto de entidades se crea al inicializar un nivel y se destruye cuando se finaliza el nivel.

También se ha definido una variable global llamada ttotal, que va ir almacenando el valor del tiempo que queda. De tal manera que al iniciar un nivel se asigna a esta variable el tiempo

total para superarlo. Mediante la acción de contador, la aplicación va descontando los segundos que van transcurriendo. Si se acaba el tiempo, mostrará la pantalla indicando que se ha acabado y si lo quiere volver a intentar. También cambia la música que se estaba escuchando en ese momento por la elegida para Game Over.

```
action contador()
{
     while( ttotal > 0 )
   {

      ttotal-=time_step/16;
      wait(1);
   }
   set(panelFin, SHOW);
   if(musica)
   {
     snd_stop(punteroMusica);
           punteroMusica = snd_play(SND_GAME_OVER,70,0);
   }
   freeze_mode =1;
}
```

Código 25: Acción contador.

Para parar el tiempo y volverlo a poner se varia el valor de una variable predefinida de 3D Game Studio que es "*freeze_mode*". Los valores que se pueden dar a esta variable son:

- "0", no hay pausa, todas las funciones corren con normalidad.
- "1", sólo se pausan las funciones de las entidades y de partículas.
- "2", se pausan todas las funciones salvo las que en su código indican que la variable proc_mode=PROC_NOFREEZE.

Esto se ha usado por ejemplo, cuando se pulsa la pausa durante el juego, en ese momento se para el tiempo y cuando vuelve a jugar se reanuda.

Capítulo 4:

Planificación y presupuesto iniciales

En esta sección del proyecto se pasa a analizar la planificación inicial, esta es un proceso básico y fundamental en todo proyecto, ya que además de valorar el tiempo que va a llevar la realización de este, permitirá a su vez estimar un presupuesto que se ajustará más a la realidad.

Para conseguir realizar una buena planificación, se divide el proyecto en diferentes fases que facilitan el cálculo de los costes y recursos necesarios, y permite hacer un seguimiento más exhaustivo, con indicadores que muestran si están cumpliendo los objetivos en el tiempo marcado.

4.1 Planificación inicial

Es necesario analizar el ciclo de vida de un videojuego ya que servirá de gran ayuda para el cálculo de la planificación inicial, ya que se detallan las diferentes etapas que se siguen en la elaboración de un videojuego.

4.1.1 Ciclo de vida de un videojuego

A nivel genérico se puede estructurar el desarrollo de un videojuego en las siguientes etapas o fases:

- Idea, en esta etapa se tiene la idea del videojuego y se le empieza a dar forma, en un proceso meramente creativo. Se definen aspectos tales como el guión del juego, género de este, modos de juego. Se hace una estimación de los costes y el tiempo que puede llevar el desarrollo del videojuego.

- Pre-Producción, en esta etapa se pasa de una fase anterior creativa a una realista, en la que se analizan factores tan importantes como si se tienen los recursos necesarios, tanto económicos como de personal cualificado, para llevarlo a cabo. Se pasa a hacer un análisis en detalle del juego y se vuelve a hacer una nueva planificación.

- Producción, en esta etapa se lleva a cabo el desarrollo del videojuego. Según se va avanzando en el desarrollo de este, se realizan pruebas por personal externo al propio desarrollo para la detección de errores y mejoras, que sirvan a los desarrolladores para mejorar el juego.

- Alfa, en este punto el juego se ha completado y ya se puede probar en su totalidad para ver de nuevo si hay errores que corregir o añadir mejoras claves.

- Beta, en esta fase sólo se busca la corrección de errores y dejar una versión estable de este.

- Liberación, en esta parte se da el visto nuevo al juego y se pasa a su distribución comercial.

- Parches, en ocasiones, una vez que sale el juego a la venta se detectan errores por parte de los usuarios que no se habían detectado en fases anteriores. Por tanto es necesario que se corrijan y se publiquen los parches para que tengan acceso a ello todo usuario de este videojuego.

- Actualizaciones, desde la entrada de los juegos online, es muy habitual que salgan actualizaciones de los videojuegos con nuevos niveles, personajes, funcionalidades... Con la finalidad de aumentar los ingresos del videojuego.

4.1.2 Ciclo de vida del proyecto

Para la realización del proyecto no se ha seguido exactamente el modelo de ciclo de vida planteado anteriormente sino que se ha optado por el "*Modelo de desarrollo evolutivo*".

Figura 75: Modelo de desarrollo evolutivo.

El desarrollo del proyecto se ha dividido en diferentes fases o módulos, siguiendo el modelo de la figurar 75, de tal manera que de cada bloque se ha hecho una toma de requisitos seguida de un diseño y su implementación. A continuación se ha presentado cada parte implementada al cliente, en este caso el tutor del proyecto, para que la evaluara y aportara mejoras o indicara correcciones porque lo implementado se alejaba de lo deseado. Cuando había algo que cambiar de un bloque, se volvía de nuevo a la fase de diseño y se repetían todos los pasos hasta la aprobación final por parte del cliente.

El proyecto se ha dividido en las siguientes fases:

- Conocimiento del motor y formación previa. Se ha tenido que hacer un estudio del funcionamiento del motor 3D Game Studio y de las diferentes aplicaciones que incorpora. En el caso de contar con personal cualificado en el uso de esta aplicación, esta fase no sería necesario.

- Documentación del proyecto. Se lleva a cabo toda la documentación del proyecto, incluida esta memoria, que se va actualizando según se va avanzando en la realización del

proyecto. También incluye toda la información útil obtenida de la anterior fase, que facilitará la fase de implementación.

- Prototipo inicial. Se genera un primer prototipo para valorar la jugabilidad del juego. En esta fase, se implementa un terreno plano con un personaje y un objeto, y se ve cómo se controla al jugador y cómo interactúa con el objeto. Se puede considerar el núcleo del juego, ya que todo este se basa en un personaje que mueve una serie de cajas dentro de un mundo 3D.

- Primer nivel. Se crea el primer nivel del juego, en el que el jugador ya puede mover todas las cajas y colocarlas en las zonas destinadas a tal efecto y el sistema detecta cuando se ha superado dicho nivel.

- Modelos de personajes. Se descargan los modelos de los personajes y se trabaja sobre ellos para que puedan moverse por el escenario y no simplemente cambiar de ubicación como se hacía en el prototipo inicial.

- Menú principal. Se diseñan e implementa el menú principal para que sea lo primero que muestre la aplicación y de acceso al primer nivel.

- Vistas del juego. Se implementan las tres vistas disponibles del juego y que el jugador puedas cambiarlas mientras está jugando la partida.

- Control del tiempo y sistema de puntuación. Se implementa el control del tiempo de partida por parte del sistema y se establece un sistema de puntuación en base al tiempo en superar el nivel.

- Guardar y cargar partida. Se implementa el sistema de guardado y carga de partidas.

- Sonido, se incorpora el sonido al sistema de juego y de navegación de menús.

- Navegación de menús. Se diseñan e implementan el resto de menús, que son el menú de pausa, opciones, cargar partida, records, pausa, y guardar partida.

- Nivel 2 y nivel 3. Se crean los dos niveles restantes.

- Pruebas. Se prueba el juego y todas las funcionalidades por completo

- Evaluación final. El cliente realiza la valoración final del proyecto.

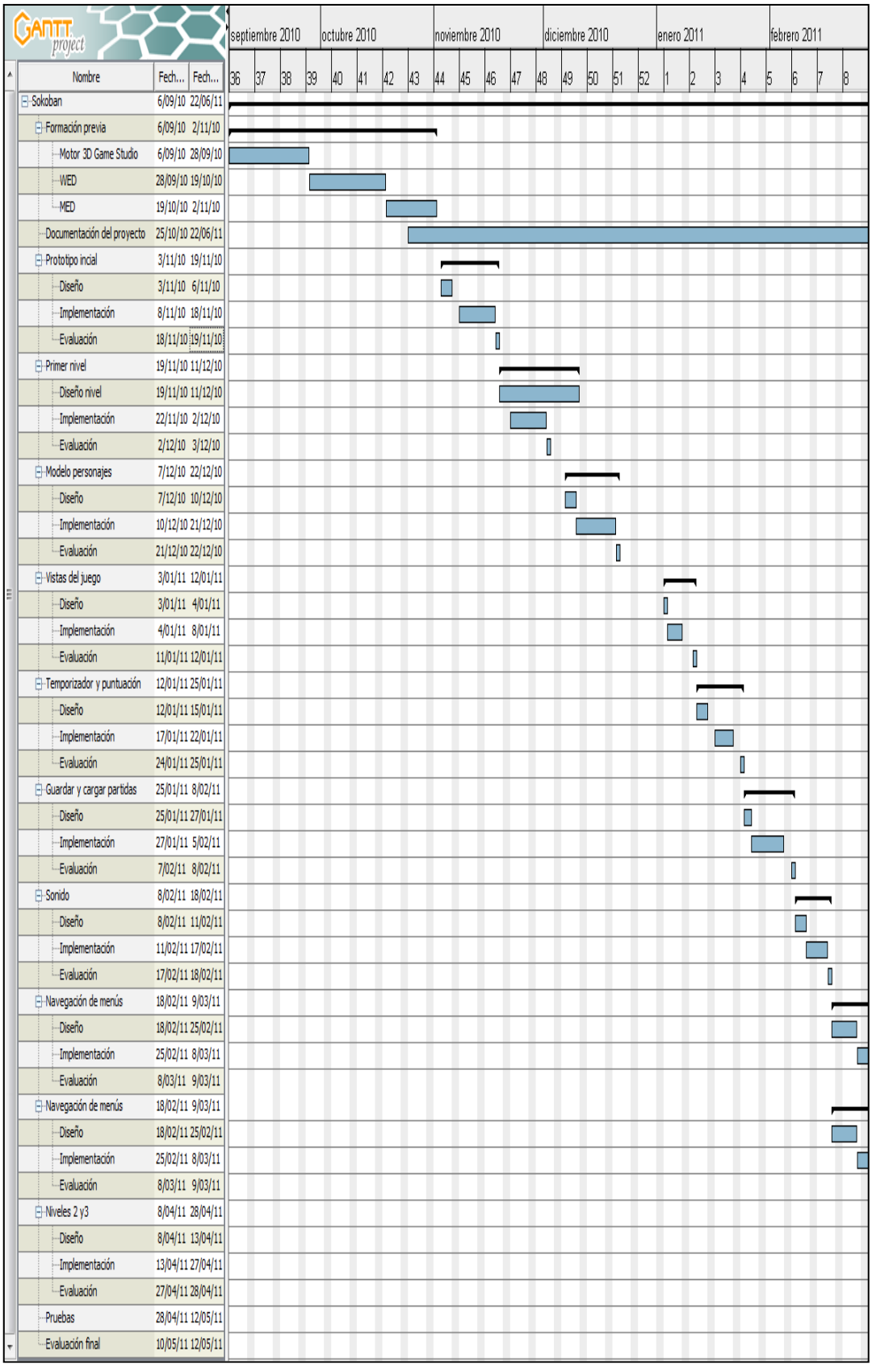

Figura 76: Diagrama de Gantt planificación inicial 1.

Nombre	Fech...	Fech...
Sokoban	6/09/10	22/06/11
Formación previa	6/09/10	2/11/10
Motor 3D Game Studio	6/09/10	28/09/10
WED	28/09/10	19/10/10
MED	19/10/10	2/11/10
Documentación del proyecto	25/10/10	22/06/11
Prototipo incial	3/11/10	19/11/10
Diseño	3/11/10	6/11/10
Implementación	8/11/10	18/11/10
Evaluación	18/11/10	19/11/10
Primer nivel	19/11/10	11/12/10
Diseño nivel	19/11/10	11/12/10
Implementación	22/11/10	2/12/10
Evaluación	2/12/10	3/12/10
Modelo personajes	7/12/10	22/12/10
Diseño	7/12/10	10/12/10
Implementación	10/12/10	21/12/10
Evaluación	21/12/10	22/12/10
Vistas del juego	3/01/11	12/01/11
Diseño	3/01/11	4/01/11
Implementación	4/01/11	8/01/11
Evaluación	11/01/11	12/01/11
Temporizador y puntuación	12/01/11	25/01/11
Diseño	12/01/11	15/01/11
Implementación	17/01/11	22/01/11
Evaluación	24/01/11	25/01/11
Guardar y cargar partidas	25/01/11	8/02/11
Diseño	25/01/11	27/01/11
Implementación	27/01/11	5/02/11
Evaluación	7/02/11	8/02/11
Sonido	8/02/11	18/02/11
Diseño	8/02/11	11/02/11
Implementación	11/02/11	17/02/11
Evaluación	17/02/11	18/02/11
Navegación de menús	18/02/11	9/03/11
Diseño	18/02/11	25/02/11
Implementación	25/02/11	8/03/11
Evaluación	8/03/11	9/03/11
Niveles 2 y3	8/04/11	28/04/11
Diseño	8/04/11	13/04/11
Implementación	13/04/11	27/04/11
Evaluación	27/04/11	28/04/11
Pruebas	28/04/11	12/05/11
Evaluación final	10/05/11	12/05/11

Figura 77: Diagrama de Gantt planificación inicial 2.

Se ha estimado según la planificación inicial, que el tiemplo que llevará la realización del proyecto será de aproximádamente unos 10 meses, desde Septiembre de 2010 hasta Junio de 2011. No se han contabilizado ni sábados ni domingos, y puesto que sólo hay un desarrollador del proyecto se han tenido en cuenta las vacaciones de este y por eso hay periodos en los que no consta que se haga ningún tipo de avance ni trabajo en el proyecto. En la siguiente tabla se pueden ver el resumen de todas las tareas:

Tarea	Inicio	Fin	Duración
Formación previa	06/09/2010	02/11/2010	41 días
Documentación del proyecto	25/10/2010	22/06/2010	172 días
Prototipo inicial	03/11/2010	19/11/2010	12 días
Primer nivel	19/11/2010	11/12/2010	16 días
Modelo personajes	07/12/2010	22/12/2010	11 días
Vistas del juego	03/01/2011	12/01/2011	7 días
Temporizador y puntuación	12/01/2011	25/01/2011	9 días
Guardar y cargar partidas	25/01/2011	08/02/2011	10 días
Sonido	08/02/2011	18/02/2011	8 días
Navegación de menús	18/02/2011	09/03/2011	13 días
Niveles 2 y 3	08/04/2011	28/04/2011	14 días
Pruebas	28/04/2011	12/05/2011	10 días
Evaluación final	10/05/2011	12/05/2011	2 días

Tabla 68: Planificación inicial del proyecto.

4.2 Presupuesto inicial

A continuación se muestra el presupuesto inicial estimado para la realización de este proyecto. El presupuesto inicial de proyecto asciende a 27.589 €, que se corresponden principalmente a los sueldos del desarrollador del proyecto y del jefe de proyecto, ya que los costes en otro tipo de recursos son bastante bajos.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Alejandro Aguilar León

2.- Departamento:
Departamento de Informática

3.- Descripción del Proyecto:

- Titulo	**Diseño e implementación del juego sokoban para pc mediante el motor 3d Game Studio**
- Duración (meses)	**10**
Tasa de costes Indirectos:	**20%**

4.- Presupuesto total del Proyecto (valores en Euros):
Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Peralta Donate, Juan		Jefe de Proyecto	1	3.100,10	3.100,10	
Aguilar León, Alejandro		Ingeniero	10	1.937,10	19.371,00	
					0,00	
					0,00	
					0,00	
		Hombres mes 11		**Total**	22.471,10	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

Figura 78: Presupuesto inicial, parte 1.

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador para desarrollo de aplicación	999,99	100	12	60	200,00
Impresora	120,00	100	12	60	24,00
Disco duro externo 1 GB	30,00	100	12	60	6,00
				Total	230,00

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Fungible	Ofistore	60,00
Viajes	-	100,00
Licencia Microsoft Office Hogar y Estudiantes	Microsoft Store	129,99
		0,00
		0,00
		0,00
		0,00
	Total	289,99

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

Figura 79: Presupuesto inicial, parte 2.

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	22.471
Amortización	230
Subcontratación de tareas	0
Costes de funcionamiento	290
Costes Indirectos	4.598
Total	27.589

Figura 80: Presupuesto inicial, parte 3.

Capítulo 5:

Conclusiones

De los objetivos planteados al inicio del proyecto se puede afirmar que se ha cumplido con todos. El objetivo principal era llegar a implementar un juego sencillo con la aplicación 3D Game Studio y a la vista de los resultados se puede concluir que se conseguido satisfactoriamente.

El segundo objetivo que nos habías marcado al inicio de este proyecto, era que este pudiera servir de guía a toda persona interesada en utilizar esta herramienta. Para cumplir con este objetivo se ha hecho un esfuerzo especial al explicar en detalle todas las fases de este. Intentando dar explicaciones sencillas y centrándonos en el valor práctico a la hora de programar y no en conceptos complejos que no harían más que complicar el entendimiento de este motor gráfico. Con lo que también consideramos que se ha llegado a cumplir satisfactoriamente.

Estos dos primeros objetivos han sido verdaderamente complejos de conseguir por la escasez de recursos, foros y manuales disponibles en la red. De hecho el único foro español que se había logrado localizar en la red era PartidaAbierta.com {94}**¡Error! No se encuentra el origen de la referencia.**, y recientemente ha anunciado que dado el poco seguimiento que tenía van a cerrar el sitio web.

El tercer objetivo marcado era dejar una documentación detallada de todo el proyecto, y a la vista de la memoria del proyecto se puede afirmar que se ha cumplido correctamente. Esta

memoria podría ser la base para alguien que estuviera interesado en profundizar más en esta herramienta y no ha llegado a manejarla previamente.

Finalmente el último objetivo marcado valorar la utilidad de este motor gráfico frente a otras disponibles en el mercado. Una vez realizado todo el proyecto llegamos a la conclusión de que no es así, por varias razones:

- La principal razón, es que aunque si bien es cierto que 3D Game Studio proporciona una versión gratuita de su motor, esta no está completa. Con lo que tiene muchas funciones inhabilitadas, que si las deseas usar tienes que comprar el motor. Entre estas funciones que no están disponibles de manera gratuita está la que te permite crear una versión ejecutable del software, con lo que no se ha podido cumplir con uno de los requisitos esenciales que tenía el proyecto. Así que para poder llegar a usar el juego que hemos implementado es necesario tener instalado 3D Game Studio en el ordenador en el que se vaya a jugar.

- Esta línea de negocio mencionada en el anterior punto, no parece una política competitiva ya que otros motores gráficos como UDK, ponen de manera gratuita todo el software disponible y sólo piden un porcentaje de ventas si la aplicación desarrollada se comercializa.

- El resultado final en cuanto a calidad gráfica de este motor es muy inferior a otros ya mencionados en la memoria.

- El editor que proporciona para la programación de scripts es muy rudimentario, y se hace realmente tedioso intentar depurar el programa con este.

- Como se ha mencionado anteriormente, son muy pocos los foros y páginas web que hablan del uso de esta herramienta. De lo que podemos sacar dos conclusiones, una que ante cualquier duda uno se encontrará solo para resolverla y que si son pocos los desarrolladores que la utilizan es porque efectivamente no debe ser una herramienta muy buena.

Tampoco todo lo que se va a decir de esta herramienta es negativo, se pueden destacar varios puntos positivos como:

- El entorno de creación de niveles WED, es muy intuitivo y sencillo de manejar y a pesar de que no se llegan a alcanzar niveles de calidad gráfica tan altos como en otros motores, nos permite llegar a implementar niveles bastante correctos.

- El lenguaje de programación que soporta, Lite-c, al ser una versión simplificada de C++ es sencillo de manejar y no tiene conceptos abstractos difíciles de entender como pueden tener otros lenguajes de programación.

- En cuanto a la animación de modelos también es muy intuitiva y fácil de manejar.

Finalmente y como una de las conclusiones más importantes para mí, es que he podido llegar a cumplir con una de las cosas que siempre había querido hacer que era llegar a crear un videojuego, que aunque ha sido un juego muy sencillo, me ha permitido hacerme una idea de todo el trabajo que hay detrás y de las diferentes partes que lo componen. También me ha permitido adquirir unos conocimientos básicos de diseño. Y refrescar y adquirir nuevos conocimientos de programación.

Capítulo 6:

Líneas futuras

Según se ha ido implementado la aplicación se iban observando partes mejorables a las que en un principio se habían planteado, pero que por el tiempo limitado para la realización del proyecto se han dejado como mejoras futuras, entre estas:

Como mejora principal, sería hacer una versión con un fichero ejecutable que permita la posibilidad de jugar en cualquier PC sin la necesidad de tener instalado el 3D Game Studio.

Otras mejoras, serían aumentar el número de niveles ya que con tan solo tres niveles el juego es excesivamente corto.

También aumentar el número de personajes disponibles y que el diseño de estos este más logrado.

Sería interesante que se pudieran jugar dos jugadores a la vez en un mismo PC con pantalla divida o jugar online. Y que se pudieran llegar a publicar los records obtenidos.

Un cambio fundamental sería dar seguridad al fichero en el que se guardan todos los datos de las partidas guardadas y puntuaciones.

En otros sokoban disponibles en la red se ha observado que cuando se hacía un movimiento erróneo se podía dar macha atrás sin necesidad de que se reiniciara el nivel.

Para mejorar el sistema de puntuación, que no sólo se basara en el tiempo sino en el número de movimientos realizados para superar el nivel.

Finalmente sería interesante ver la posibilidad de llegar a convertir este juego en un juego multiplataforma, pero esto conllevaría usar otro motor gráfico.

Capítulo 7:

Referencias

En este capítulo se incluyen todas las referencias a documentos y páginas webs que se han utilizado tanto para el desarrollo del proyecto, como para la creación de este manual.

- [1] Paul Heydon. Disponible [Internet]: <http://www.avistapartners.com/people-paul.html> [31 de Mayo de 2011].
- [2] XBOXGO. Disponible [Internet]: <http://xboxgo.es/14288/la-industria-del-videojuego-genera-105-000-millones-de-dolares > [31 de Mayo de 2011].
- [3] Motor grafico. Definición de motores gráficos. Disponible [Internet]: <http://es.wikipedia.org/wiki/Motor_de_videojuego> [31 de Mayo de 2011].
- [4] 3D Game Studio A7. Disponible [Internet]: <http://www.conitec.com/english/gstudio/index.php> [06 de Junio de 2011].
- [5] *Sokoban.* Disponible [Internet]: <http://games4dos.ro/?m=201008&paged=3> [31 de Mayo de 2011].
- [6] Spectrum HoloByte. Disponible [Internet]: <http://en.wikipedia.org/wiki/Spectrum_HoloByte> [31 de Mayo de 2011].
- [7] 3D Game Studio. Tutorial Online. Disponible [Internet]: <http://tutorial.3dGame Studio.net> [27 de Junio de 2011].
- [8] 3D Game Studio. Tutorial Online. Disponible [Internet]: <http://manual.3dGame Studio.net/> [27 de Junio de 2011].

- [9] Descarga 3D Game Studio A7. Disponible [Internet]: <http://server.conitec.net/down/gstudio7_setup.exe> [27 de Junio de 2011].
- [10] Eezpix. Descarga eezpix. Disponible [Internet]: < http://eezpix.softonic.com/> [27 de Junio de 2011].
- [11] Tomas T. Goldsmith. Biografía Disponible [Internet]: <http://www.nytimes.com/2009/03/15/arts/television/15goldsmith.html> [27 de Junio de 2011].
- [12] Orígenes de los videojuegos. Disponible [Internet]: <http://indicelatino.com/juegos/historia/origenes/> [27 de Junio de 2011].
- [13] Tres en raya 1952. Video de tres en raya. Disponible [Internet]: <http://www.youtube.com/watch?v=Xyi0_ViYmFY&feature=related> [27 de Junio de 2011].
- [14] Tennis for two. Disponible [Internet]: <http://www.pixfans.com/tennis-for-two-el-primer-juego-electronico-de-la-historia/> [27 de Junio de 2011].
- [15] Space War. Video Space War de Steve Russell. Disponible [Internet]: <http://www.youtube.com/watch?v=Rmvb4Hktv7U> [27 de Junio de 2011].
- [16] PDP-1. Disponible [Internet]: <http://es.wikipedia.org/wiki/PDP-1> [27 de Junio de 2011].
- [17] Ralph Baer. Página oficial. Disponible [Internet]: <http://www.ralphbaer.com/> [27 de Junio de 2011].
- [18] Magnavox. Disponible [Internet]: <http://es.wikipedia.org/wiki/Magnavox> [27 de Junio de 2011].
- [19] Magnavox-odyssey. Disponible [Internet]: <http://www.magnavox-odyssey.com/> [27 de Junio de 2011].
- [20] Nolan Bushnell. Disponible [Internet]: <http://es.wikipedia.org/wiki/Nolan_Bushnell> [27 de Junio de 2011].
- [21] Nutting_Associates. Disponible [Internet]: <http://en.wikipedia.org/wiki/Nutting_Associates> [27 de Junio de 2011].

- [22] Computer Space. Video demostración Computer Space. Disponible [Internet]: <http://www.youtube.com/watch?v=b3BQsCCwo8w> [27 de Junio de 2011].
- [23] Atari. Atari según la Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Atari> [27 de Junio de 2011].
- [24] Pong. Video de Pong. Disponible [Internet]: <http://www.youtube.com/watch?v=LPkUvfL8T1I> [13 de Junio de 2011].
- [25] Taito. Página oficial. Disponible [Internet]: <http://www.taito.com/> [13 de Junio de 2011].
- [26] Tank. Video de Tank. Disponible [Internet]: <http://www.youtube.com/watch?v=3OsBUzYBJgU> [13 de Junio de 2011].
- [27] Space Race. Video de Space Race. Disponible [Internet]: <http://www.youtube.com/watch?v=mB_1N9tUjcM> [13 de Junio de 2011].
- [28] TTL. Disponible [Internet]: <http://es.wikipedia.org/wiki/Tecnolog%C3%ADa_TTL> [13 de Junio de 2011].
- [29] Space Invaders. Video de Space Invaders. Disponible [Internet]: <http://www.youtube.com/watch?v=5xqDW7k_e40&feature=fvst> [13 de Junio de 2011].
- [30] Pacman. Video de Pacman. Disponible [Internet]: <http://www.youtube.com/watch?v=33vjB-Vtevg> [13 de Junio de 2011].
- [31] Nintendo. Web oficial de Nintendo. Disponible [Internet]: <http://www.nintendo.es/> [13 de Junio de 2011].
- [32] Game and watch. Disponible [Internet]: <http://es.wikipedia.org/wiki/Game_&_Watch> [13 de Junio de 2011].
- [33] Sinclair ZX Spectrum. Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Sinclair_ZX_Spectrum> [13 de Junio de 2011].
- [34] Comodore 64. Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Commodore_64> [13 de Junio de 2011].
- [35] Amstrad CPC. Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Amstrad_CPC> [13 de Junio de 2011].

- [36] Hiroshi Yamauchi. Biografía en N-Sider Disponible [Internet]: <http://www.n-sider.com/contentview.php?contentid=224> [13 de Junio de 2011].
- [37] Shigeru Miyamoto. Wikipedia. Disponible [Interne]: <http://es.wikipedia.org/wiki/Shigeru_Miyamoto> [13 de Junio de 2011]
- [38] NES. NES según Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/NES> [13 de Junio de 2011].
- [39] Video NES. Video demostración de NES. Disponible [Internet]: <http://www.youtube.com/watch?v=WYjgHLFZMa0&feature=player_embedded#at=16> [13 de Junio de 2011].
- [40] Super Mario Bros. Video de Super Mario Bros. Disponible [Internet]: <http://www.youtube.com/watch?v=xkD7L2QFwR0> [13 de Junio de 2011].
- [41] Sega. Web oficial de Sega. Disponible [Internet]: <http://www.sega.com/> [13 de Junio de 2011].
- [42] Sega Master System. Según Wikipedia. Disponible [Internet]: < http://es.wikipedia.org/wiki/Sega_Master_System> [13 de Junio de 2011].
- [43] Alex Kidd. Video de Alex Kidd. Disponible [Internet]: <http://www.youtube.com/watch?v=yZaKPVpEvZQ> [13 de Junio de 2011].
- [44] Mega Drive. Según Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Sega_Mega_Drive> [13 de Junio de 2011].
- [45] Sonic The Hedgehog. Video de Sonic. Disponible [Internet]: <http://www.youtube.com/watch?v=CqOlpQ7sepE> [13 de Junio de 2011].
- [46] Game Boy. Según Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Game_Boy> [13 de Junio de 2011].
- [47] Tetris. Video de Tetris. Disponible [Internet]: <http://www.youtube.com/watch?v=AaGqU2pUFqA> [13 de Junio de 2011].
- [48] Supernintendo. Disponible [Internet]: <http://www.nintendo.es/NOE/es_ES/systems/super_nintendo_1110.html> [13 de Junio de 2011].

- [49] Super Mario Kart. Video de Super Mario Kart. Disponible [Internet]: <http://www.youtube.com/watch?v=HVi8AiwKn3k> [13 de Junio de 2011].
- [50] Sega Saturn. Según Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Sega_Saturn> [13 de Junio de 2011].
- [51] Virtua Fighter. Video de Virtua Fighter. Disponible [Internet]: <http://www.youtube.com/watch?v=Z6Q9DGMXFEA> [12 de Junio de 2011].
- [52] Nintendo 64. Según Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Nintendo_64> [13 de Junio de 2011].
- [53] Sony. Web oficial de Sony. Disponible [Internet]: <http://www.sony.es/section/home> [13 de Junio de 2011].
- [54]PlayStation. Web oficial de PlayStation. Disponible [Internet]: <http://es.playstation.com/> [13 de Junio de 2011].
- [55] Resident Evil 1. Video de Residen Evil 1. Disponible [Internet]: <http://www.youtube.com/watch?v=lyZF-vyyFn4> [13 de Junio de 2011].
- [56] PlayStation2. Según Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Playstation_2> [13 de Junio de 2011].
- [57] Emotion Engine. Según Wikipedia. Disponible [Internet]: <http://en.wikipedia.org/wiki/Emotion_Engine> [13 de Junio de 2011].
- [58] Resident Evil 4. Video de Resident Evil 4. Disponible [Internet]: <http://www.youtube.com/watch?v=mYdPCLIjLL8> [13 de Junio de 2011].
- [59] FIFA 2007. Video de FIFA 2007. Disponible [Internet]: <http://www.youtube.com/watch?v=zkg9tnecEUo> [13 de Junio de 2011].
- [60] ProEvolution Soccer 2007. Video de PES 2007. Disponible [Internet]: <http://www.youtube.com/watch?v=GQuLcg0hs98> [13 de Junio de 2011].
- [61] Microsoft. Web oficial de Microsoft. Disponible [Internet]: <http://www.microsoft.com/es/es/default.aspx> [13 de Junio de 2011].
- [62] Xbox. Web oficial de Xbox. Disponible [Internet]: <http://www.xbox.com/es-ES> [13 de Junio de 2011].
- [63] Intel Pentium III. Según wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Intel_Pentium_III> [13 de Junio de 2011].

- [64] Halo. Video de Halo. Disponible [Internet]: <http://www.youtube.com/watch?v=Kjgket__rmY&feature=related> [13 de Junio de 2011].
- [65] Xbox 360. Según Wikipedia. Disponible [Internet]: <http://en.wikipedia.org/wiki/Xbox360> [13 de Junio de 2011].
- [66] Kinect. Video de Kinect Sports. Disponible [Internet]: <http://www.youtube.com/watch?v=rYPYPCWIFKc&feature=fvwrel> [13 de Junio de 2011].
- [67] PS3. Según wikipedia. Disponible [Internet]: <http://en.wikipedia.org/wiki/Playstation_3> [13 de Junio de 2011].
- [68] Metal Gear Solid 4. Video de MGS4. Disponible [Internet]: <http://www.youtube.com/watch?v=P6V0JDyHoAU> [13 de Junio de 2011].
- [69] Wii. Según wikipedia. Disponible [Internet]: <http://en.wikipedia.org/wiki/Wii> [13 de Junio de 2011].
- [70] Wii Sports. Video de wii sports. Disponible [Internet]: <http://www.youtube.com/watch?v=zqaPFAZS1K8> [13 de Junio de 2011].
- [71] Abadía Digital. Disponible [Internet]: <http://www.abadiadigital.com/articulo/las-consolas-mas-vendidas-de-la-historia/>[13 de Junio de 2011].
- [72] Demostración Sokoban. Video de Sokoban original. Disponible [Internet]: <http://www.youtube.com/watch?v=i_WmVBwEE4U> [13 de Junio de 2011].
- [73] Boxxle. Video de Boxxle. Disponible [Internet]: <http://www.youtube.com/watch?v=6dd6IF-wJxU> [13 de Junio de 2011].
- [74] Sokoban DS. Video de Sokoban DS. Disponible [Internet]: <http://www.youtube.com/watch?v=sAIzkQJO_uI> [13 de Junio de 2011].
- [75] Sokoban Online. Enlaces a páginas con diferentes versiones de Sokoban. Disponible [Internet]: <http://www.abelmartin.com/rj/sokoban2.html>[13 de Junio de 2011].
- [76] Magic Crates. Juego Magic Crates. Disponible [Internet]: <http://www.abelmartin.com/rj/sokoban/magic_crates.html> [13 de Junio de 2011].

- [77] Direct 3D. Según wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Direct3D>[13 de Junio de 2011].
- [78] Neoteo. Historia de los videojuegos. Disponible [Internet]: <http://www.neoteo.com/top-10-los-motores-graficos-mas-importantes> [16 de Junio de 2011].
- [79] Unreal Engine. Según wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Unreal_Engine_3#Unreal_Engine_3> [16 de Junio de 2011]
- [80] Epic Games. Web oficial de Epic Games. Disponible [Internet]: <http://www.epicgames.com/> [16 de Junio de 2011].
- [81] Gears of war. Video de Gears of war. Disponible [Internet]: <http://www.youtube.com/watch?v=ccWrbGEFgI8>. [16 de Junio de 2011].
- [82] CryEngine. Web oficial de CryEngine Disponible [Internet]: <http://mycryengine.com/> [16 de Junio de 2011].
- [83] Crysis 2. Video de Crysis 2. Disponible [Internet]: <http://www.youtube.com/watch?v=laIghwTFdqg>. [16 de Junio de 2011].
- [84] Ubisoft. Web oficial de Ubisoft. Disponible [Internet]: <http://www.ubi.com/es/default.aspx> [16 de Junio de 2011].
- [85] Assassin's Creed. Video de Assassin's Creed. Disponible [Internet]: <http://www.youtube.com/watch?v=kXu3pdxeKpQ> [16 de Junio de 2011].
- [86] Microsoft XNA. Disponible [Internet]: <http://es.wikipedia.org/wiki/Microsoft_XNA> [16 de Junio de 2011].
- [87] Storage. Web oficial del juego Storage. Disponible [Internet]: <http://metocas.com/storage-728.shtml> [16 de Junio de 2011].
- [88] Resco-sokoban. Disponible [Internet]: <http://www.zonapda.net/archivo/2006/05/15/resco-sokoban/> [16 de Junio de 2011].
- [89] Rokoban. Disponible [Internet]: <http://zalods.blogspot.com/2008/07/rokoban-sokoban-version-in-3d.html> [16 de Junio de 2011].

- [90] Renderizar. Definición de renderizar según la wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Renderizar>. [8 de Julio de 2011].
- [91] 3ds *Max*. Disponible [Internet]: <http://www.autodesk.es/adsk/servlet/pc/index?siteID=455755&id=14626995> [13 de Junio de 2011].
- [92] Acnex Unlimited Resources. Disponible [Internet]: <http://au.conitec.net/> [13 de Junio de 2011].
- [93] VGMusic. Web de descarga de sonido de videojuegos clásicos. Disponible [Internet]: <http://www.vgmusic.com/> [13 de Junio de 2011].
- [94]PartidaAbierta.com. Foro de desarrolladores de 3D Game Studio. Disponible [Internet]: <http://www.partidabierta.com> [16 de Junio de 2011]

Anexo A:

Planificación final y conclusiones

En este anexo se mostraran la planificación y presupuesto final del proyecto y se hará un breve estudio de las desviaciones que han sucedido respecto de las estimaciones iniciales.

A.1 Planificación final

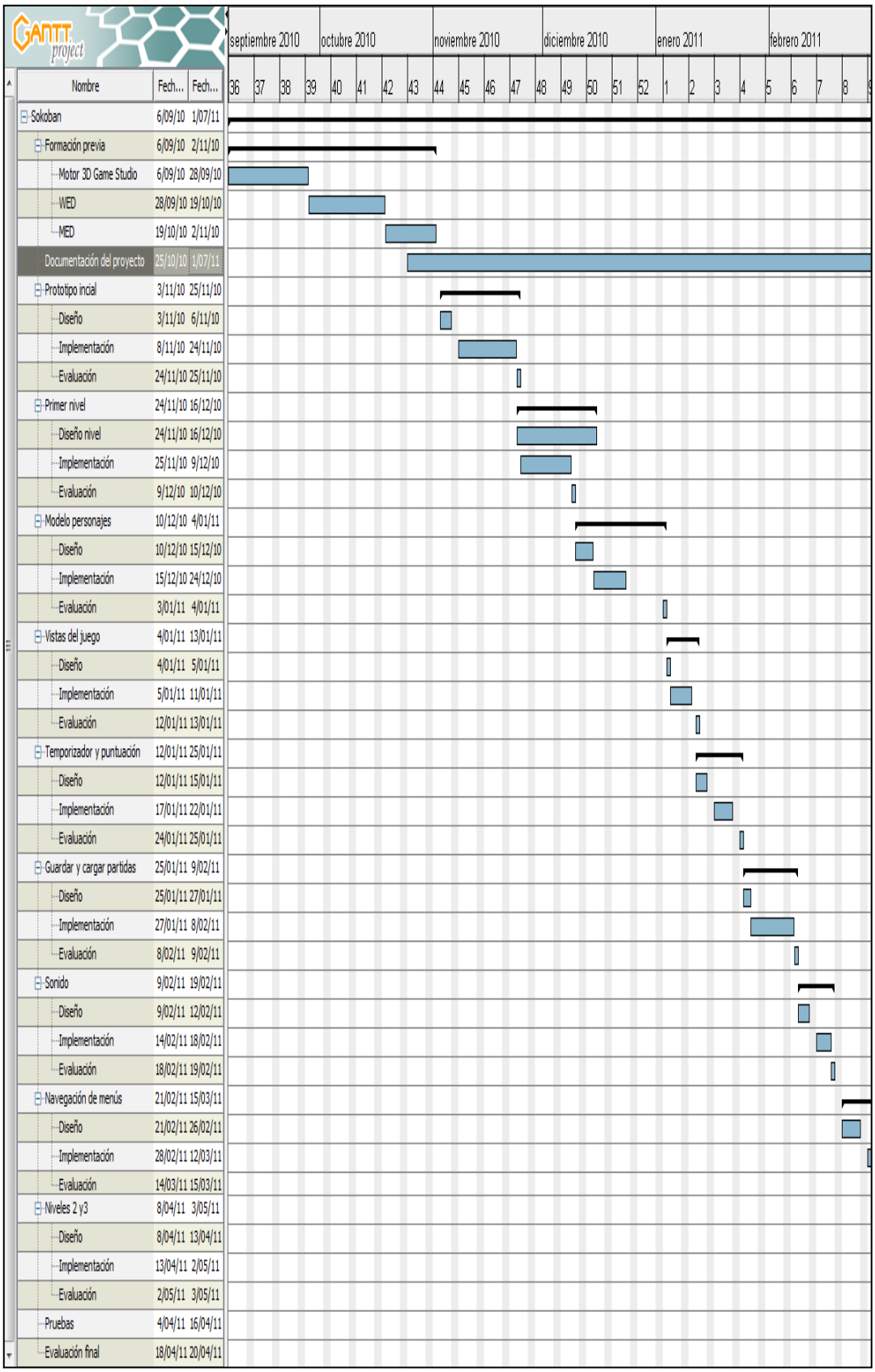

Figura 81: Diagrama de Gantt planificación final 1.

Nombre	Fech...	Fech...
Sokoban	6/09/10	1/07/11
Formación previa	6/09/10	2/11/10
Motor 3D Game Studio	6/09/10	28/09/10
WED	28/09/10	19/10/10
MED	19/10/10	2/11/10
Documentación del proyecto	25/10/10	1/07/11
Prototipo incial	3/11/10	25/11/10
Diseño	3/11/10	6/11/10
Implementación	8/11/10	24/11/10
Evaluación	24/11/10	25/11/10
Primer nivel	24/11/10	16/12/10
Diseño nivel	24/11/10	16/12/10
Implementación	25/11/10	9/12/10
Evaluación	9/12/10	10/12/10
Modelo personajes	10/12/10	4/01/11
Diseño	10/12/10	15/12/10
Implementación	15/12/10	24/12/10
Evaluación	3/01/11	4/01/11
Vistas del juego	4/01/11	13/01/11
Diseño	4/01/11	5/01/11
Implementación	5/01/11	11/01/11
Evaluación	12/01/11	13/01/11
Temporizador y puntuación	12/01/11	25/01/11
Diseño	12/01/11	15/01/11
Implementación	17/01/11	22/01/11
Evaluación	24/01/11	25/01/11
Guardar y cargar partidas	25/01/11	9/02/11
Diseño	25/01/11	27/01/11
Implementación	27/01/11	8/02/11
Evaluación	8/02/11	9/02/11
Sonido	9/02/11	19/02/11
Diseño	9/02/11	12/02/11
Implementación	14/02/11	18/02/11
Evaluación	18/02/11	19/02/11
Navegación de menús	21/02/11	15/03/11
Diseño	21/02/11	26/02/11
Implementación	28/02/11	12/03/11
Evaluación	14/03/11	15/03/11
Niveles 2 y3	8/04/11	3/05/11
Diseño	8/04/11	13/04/11
Implementación	13/04/11	2/05/11
Evaluación	2/05/11	3/05/11
Pruebas	4/04/11	16/04/11
Evaluación final	18/04/11	20/04/11

Figura 82: Diagrama de Gantt planificación final 2.

En la siguiente tabla se pueden ver el resumen del tiempo que han llevado finalmente el desarrollo de todas las tareas:

Tarea	Inicio	Fin	Duración
Formación previa	06/09/2010	02/11/2010	41 días
Documentación del proyecto	25/10/2010	22/06/2010	179 días
Prototipo inicial	03/11/2010	19/11/2010	16 días
Primer nivel	19/11/2010	11/12/2010	16 días
Modelo personajes	07/12/2010	22/12/2010	17 días
Vistas del juego	03/01/2011	12/01/2011	7 días
Temporizador y puntuación	12/01/2011	25/01/2011	9 días
Guardar y cargar partidas	25/01/2011	08/02/2011	11 días
Sonido	08/02/2011	18/02/2011	8 días
Navegación de menús	18/02/2011	09/03/2011	16 días
Niveles 2 y 3	08/04/2011	28/04/2011	17 días
Pruebas	28/04/2011	12/05/2011	10 días
Evaluación final	10/05/2011	12/05/2011	2 días

Tabla 69: Planificación final del proyecto.

A.2 Conclusiones planificación

Como se puede observar en la gráfica que aparece en la figura 83, no ha habido grandes diferencias entre la planificación inicial y final. Ha habido tres tareas que se han salido de la planificación inicial, en concreto, la tarea del prototipo inicial, creación del primer nivel y navegación de menús. Esta desviación se ha debido que han surgido varias dudas y cuestiones que ante la escasez de documentación e información a la que acudir en la red, han sido costosas de resolver. Por otro lado a pesar de estos retrasos, el día final de entrega no ha alejado de la previsión de inicial ya que se han disfrutado de menos días de vacaciones para poder cumplir con los objetivos marcados.

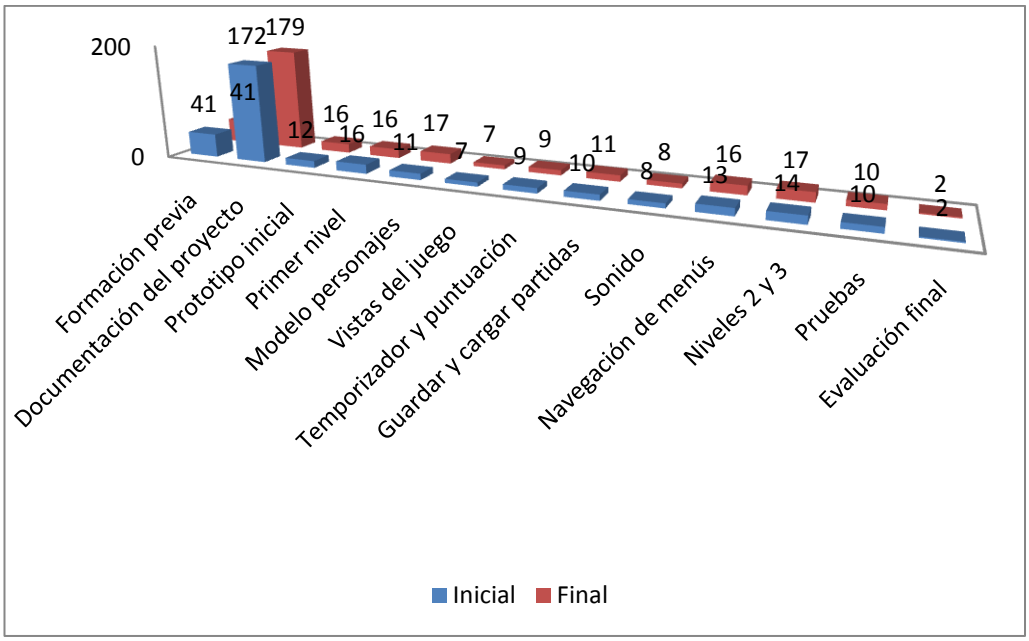

Figura 83: Comparación de planificación inicial y final.

A.3 Presupuesto final

Fruto de los retrasos anteriormente mencionados y que ante las dificultades encontradas a la hora de resolver dudas en la realización del proyecto, el presupuesto final ha sufrido unos ligeros cambios. Estos cambios se pueden resumir en dos puntos, el primero 15 días más de sueldo del desarrollador y un mes más de sueldo del jefe de proyecto, lo que ha supuesto un encarecimiento del proyecto de 4883€.

A continuación se muestra el presupuesto final del proyecto.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Alejandro Aguilar León

2.- Departamento:
Departamento de Informática

3.- Descripción del Proyecto:

- Titulo	**Diseño e implementación del juego sokoban para pc mediante el motor 3d Game Studio**
- Duración (meses)	**10,5**
Tasa de costes Indirectos:	20%

4.- Presupuesto total del Proyecto (valores en Euros):
32.471,69 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Peralta Donate, Juan		Jefe de Proyecto	2	3.100,10	6.200,20	
Aguilar León, Alejandro		Ingeniero	10,5	1.937,10	20.339,55	
					0,00	
					0,00	
					0,00	
		Hombres mes 12,5		**Total**	26.539,75	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

Figura 84: Presupuesto final, parte 1.

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador para desarrollo de aplicación	999,99	100	12	60	200,00
Impresora	120,00	100	12	60	24,00
Disco duro externo 1 GB	30,00	100	12	60	6,00
				Total	230,00

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Fungible	Ofistore	60,00
Viajes	-	100,00
Licencia Microsoft Office Hogar y Estudiantes	Microsoft Store	129,99
		0,00
		0,00
		0,00
		0,00
	Total	289,99

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

Figura 85: Presupuesto final, parte 2.

6.- Resumen de costes

Presupuesto Costes Totales	Presupuesto Costes Totales
Personal	26.540
Amortización	230
Subcontratación de tareas	0
Costes de funcionamiento	290
Costes Indirectos	5.412
Total	32.472

Figura 86: Presupuesto inicial, parte 3.